UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR
Dpto. de INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

GRADO EN INGENIERÍA ELECTRÓNICA
INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

CREACIÓN DEL MODELO DEL ROBOT HUMANOIDE TEO PARA EL SIMULADOR WEBOTS.

Autor: Jinhua Weng

Profesor: Santiago Martínez de la casa Díaz

Fecha: 05/09/2012

Resumen

En la actualidad los robots con desplazamiento mediante ruedas son más eficientes y dan mayor velocidad a las tareas programadas, pero la idea es conseguir robots bípedos para realizar tareas más versátiles.

El Departamento de Ingeniería de Sistemas y Automática de nuestra universidad (UC3M) está investigando y construyendo un robot humanoide real del modelo TEO. Para ello, se necesita construir un modelo en el simulador Webots.

El objetivo de este trabajo fin de grado es la creación de dicho modelo para el simulador Webots a partir del modelo tridimensional existente (robot humanoide TEO). Dicho modelado permitirá la simulación de tareas del Robot Humanoide TEO en el entorno virtual, permitiendo comprobar las características necesarias para el movimiento del mismo.

ABSTRACT

At present, the robots with wheels are more efficient and faster to do program tasks. But the idea is that human robots can perform the work more versatile.

The Department of Systems Engineering and Automation of our university (UC3M) is researching and building a real human robot model TEO. For it, we need to build a model in Webots simulator.

The aim of this project is to create the model for the Webots simulator from the existing three-dimensional model (human robot TEO). This model allows the simulation of the human robot TEO in the virtual environment, and it can also check the necessary characteristics for its movement.

Índice

Índice de figuras:

Índice de tablas:

1 Introducción

1.1 Introducción

Un robot es una máquina o ingenio electrónico programable capaz de realizar tareas repetitivas de forma más rápida, barata y precisa que los seres humanos, pudiendo además interaccionar con su entorno. La robótica humanoide es el área de la ingeniería dedicada al desarrollo de sistemas robotizados que buscan imitar las peculiaridades del ser humano.

Buscando imitar al ser humano en aspecto, el robot humanoide estará compuesto por dos piernas que le proporcionarán el movimiento, dos brazos que le ayudarán a equilibrarse o sujetarse y por una cabeza que, aunque no sea imprescindible, le aportará un aspecto más humano.

La principal ventaja de los robots humanoides sobre otro tipo de robots es que pueden trabajar directamente en el mismo entorno que los humanos, sin que se deban realizar modificaciones sobre dicho entorno, sin embargo, la complejidad en el diseño y control aumentan considerablemente. Debido a su complejidad de diseño y construcción, dichos autómatas son muy caros y delicados ya que se pueden dañar fácilmente. Por esta razón es conveniente proporcionar a los usuarios un simulador de robots que les proporcione un entorno donde poder realizar pruebas sin riesgo de daño sobre los autómatas. En este proyecto se utilizara el simulador Webots para crear el modelo del robot humanoide TEO.

1.2 Motivación

En la actualidad existen muchas áreas de investigación en el campo de la robótica humanoide. Todavía no existen en el mercado robots fiables y robustos de tamaño humano, y la bipedestación todavía no está resuelta completamente.

El Departamento de Ingeniería de Sistemas y Automática de nuestra universidad está investigando y construyendo un robot humanoide real del modelo TEO. Para ello, se necesita construir un modelo en el simulador Webots y posteriormente simularán algunos tipos de movimiento para comprobar su correcto funcionamiento.

En la actualidad los robots con desplazamiento mediante ruedas son más eficientes y dan mayor velocidad a las tareas programadas, pero la idea es conseguir robots bípedos para realizar tareas más versátiles y poder desplazarse en terrenos irregulares.

La finalidad de este tipo de robots es el acercamiento a tareas propias del ser humano y poder relacionarse con ellos con mayor facilidad.

Este simulador es un programa que permite al usuario crear y modificar un robot manipulador que imita el comportamiento del ser humano. Las ventajas de la simulación son entre otras la posibilidad de evaluar su posible desempeño y realizar infinidad de pruebas ahorrando muchos recursos, costes y tiempo.

El modelo de este proyecto podría servir como plataforma para la prueba de movimientos de robots de tipo humanoide.

1.3 Objetivos

El objetivo de este proyecto es la creación del modelo del Robot Humanoide TEO para el simulador Webots a partir del modelo tridimensional existente. Dicho modelado permitirá la simulación de tareas del Robot Humanoide TEO en el entorno virtual, permitiendo comprobar las características necesarias para el movimiento del mismo.

En primer lugar, se debe realizar un estudio exhaustivo del simulador Webots y sus características principales con el objetivo de familiarizarse con el entorno de trabajo.

A continuación, se desarrollará el modelo del robot en la misma plataforma Webots siguiendo todas las características cinemáticas del robot humanoide TEO.

Una vez desarrollado el modelo se simularán algunos tipos de movimiento como por ejemplo: levantar el brazo, levantar la pierna, imitar la forma de caminar de los humanos, etc. Para ello, construimos un controlador y utilizamos el lenguaje C para programar las secuencias necesarias.

Posteriormente, se comprobarán dichas tareas (ejecución, rendimiento, etc.) y el modelo del robot en el simulador Webots.

Por último, se debe grabar los videos de todas las secuencias de sus movimientos, y la posibilidad de extracción de datos como por ejemplo: la extracción del modelo en lenguaje VRML, la extracción de controlador, etc.

2 Robot Humanoide

2.1 Historia de la robótica

La palabra "robot", es de origen checo y significa siervo o esclavo; fue inventada por el escritor checo Karel Capek (1890-1938) en su obra teatral R.U.R., estrenada en Europa en 1920. [1]

El ser humano lleva siglos soñando con la creación de máquinas autómatas y obedientes, capaces de llevar a cabo los trabajos duros y repetitivos que no requieran capacidad de improvisación.

El inicio de la robótica actual puede fijarse en la industrial textil del siglo XVIII, cuando Joseph Jacquard inventó en 1801 una maquina textil programable mediante tarjetas perforadas. Luego, la Revolución Industrial impulsó el desarrollo de estos agentes mecánicos.

En 1805, Henri Maillardert construyó una muñeca mecánica que era capaz de hacer dibujos, una serie de levas se utilizaban como “programa” para el dispositivo durante el proceso de escribir y dibujar.

La Revolución Industrial y la creación del sistema de cadenas de montaje, que divide la fabricación de cualquier elemento en pequeñas tareas, fue el primer gran paso hacia el logro de la robotización total en las industrias.

El concepto popular de un robot es el de aquel elemento de apariencia humana que actúa como tal. Este concepto de humanoide ha sido inspirado y estimulado por varias narraciones de ciencia ficción.

Actualmente, se están desarrollando robots altamente inteligentes con más y mejores extensiones sensoriales, para entender sus acciones y captar el mundo que los rodea. Esto permite una toma inteligente de decisiones y el control del proceso en tiempo real.

2.2 Conceptos básicos

La definición adoptada por el Instituto Norteamericano de Robótica aceptada internacionalmente para Robot es: *Manipulador multifuncional y reprogramable, diseñado para mover materiales, piezas, herramientas o dispositivos especiales, mediante movimientos programados y variables que permiten llevar a cabo diversas tareas.*[1]

Un Robot es un dispositivo comúnmente mecánico, creado por el ser humano, que suele desempeñar tareas automáticamente gracias a la programación llevada a cabo por las personas.

En 1939 el autor de ciencia ficción ***Isaac Asimov*** propuso tres leyes fundamentales de la robótica. Son las reglas de comportamiento que deberán respetar los robots cuando sean lo bastante evolucionados para vivir entre los hombres y capaces de tener razonamientos abstractos. Estas leyes surgen como medida de protección para los seres humanos. [1]

> **Primera ley:** Un robot no puede hacer daño a un ser humano o, por inacción, permitir que un ser humano sufra daño.
>
> **Segunda ley:** Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si entrasen en conflicto con la Primera Ley.
>
> **Tercera ley:** Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la Primera o la Segunda Ley.

Por otra parte, los robots suelen ser reprogramables y multifuncionales, con conexión de retroalimentación y cuya inteligencia viene dada por una computadora o un microcontrolador que ejecuta un programa. Sin embargo se han desarrollado mucho los Robots con inteligencia alámbrica, cuyas acciones son generalmente llevadas a cabo por máquinas que mueven extremidades o impulsan al robot.

2.3 Actualidad en robots humanoides

El objetivo principal de algunos investigadores en robótica es construir robots que se parezcan a las personas, tanto en su cuerpo como en su comportamiento. Sin embargo, hasta ahora, los robots más utilizados en investigaciones robóticas han sido los robots manipuladores, los móviles y los robots con más de 2 patas.

La característica básica de robot humanoide está compuesto por dos piernas que le proporcionaran el movimiento, por dos brazos que le ayudaran a equilibrarse o a sujetarse y por una cabeza que aunque no sea imprescindible, le aportará un aspecto más humano.

La principal ventaja de los robots humanoide sobre otro tipo de robots es que pueden trabajar directamente en el mismo entorno que los humanos, otro aspecto importante es que el medio de locomoción de los robots humanoides se puede adaptar a su entorno, ya que la mayoría de utensilios, maquinarias y escenarios (oficina, vivienda, metro, cine, calle, etc.) están adaptados para el uso humano, también estarán para el uso de robots humanoides.

Respecto al estado actual de las investigaciones en robots humanoides he de destacar que:

- La locomoción bípeda y estable no está resuelta totalmente.
- No existe robots humanoides de tamaño humano en el mercado, ya que la complejidad del problema de una locomoción bípeda y estable crece exponencialmente con la altura y peso del humanoide. Por lo tanto, predominan los humanoides pequeños, ya que pueden llevar a cabo un movimiento de caminar complejo de forma más sencilla y, además, son más baratos y más fáciles de controlar.

A continuación se presenta una lista de los mejores robots humanoides:[2]

2.3.1 Nao Robot

Nao, un robot humanoide increíble desarrollado por la empresa francesa Aldebaran robotics, que en su última versión de marzo 2008, llamado ***Nao Robocup Edition (ver figura 1 y 2)***, muestra muchísimas cualidades motrices y una gran interactividad con el entorno. Muestra los grandes avances en el desarrollo de robots humanoides, donde no solo puede hacer tareas repetitivas sino que puede realizar muchísimas tareas dependiendo el entorno

Figura 1 Nao Robot

y que se puede adaptar a nuestras necesidades ya que es programable.

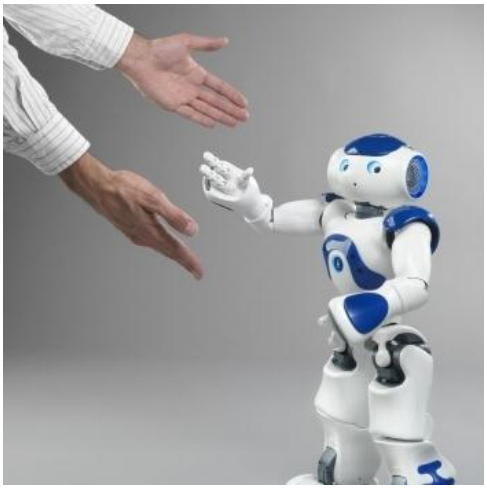

Figura 2 Nao Robot

Entre sus cualidades principales se destacan su reconocimiento de voz y de órdenes, puede detectar las diferentes formas de los objetos, rostros y seguir su moviendo, es sensible al tacto en muchas partes de su cuerpo, tiene conectividad Wi-Fi que inclusive le permite comunicarse con otros robots de su misma clase, entre muchos otro. Pero en vez de nombrar sus cualidades, que mejor que ver sus capacidades en acción, en los siguientes videos vemos muchas de sus grandes características robóticas.

2.3.2 NEW ASIMO

ASIMO, es más famoso de los robots humanoides creado por Honda, que desde el principio mostró grandes cualidades para la translación bípeda (ver figura 3). Fue el que dio el gran primer paso en lograr que un robot caminara y es uno de los mejores robots de la actualidad.

Figura 3 ASIMO

Este robot tiene 5 capacidades principales, y son las siguientes:

1. Reconocimiento de objetos en movimiento. Entre las funcionalidades más importantes se incluye la capacidad de seguir los movimientos de las personas con su cámara, para seguir a una persona, o saludar a una persona cuando él o ella se acerca.
2. Reconocimiento de las posturas y los gestos. Debido a esto puede reaccionar y además de poder ser dirigido por comandos de voz, también sigue los movimientos naturales de los seres humanos. Esto permite, por ejemplo, reconocer cuando se ofrece un saludo de manos.
3. Reconocimiento de Medio Ambiente. Esto sirve, por ejemplo, para detectar los peligros potenciales, tales como escaleras, y evitar golpear a los seres humanos u otros objetos en movimiento.
4. Distinguir los sonidos. Puede responder muchas preguntas, ya sea por un breve movimiento del cuerpo en general o de solo la cabeza, o una respuesta verbal.

5. El reconocimiento del rostro. Se puede reconocer de forma individual aproximadamente 10 caras diferentes. Una vez que estén guardados en su memoria puede responderles por su nombre.

2.3.3 HRP-4C

Figura 4 HRP-4C

HRP-4C es una de los últimos robots considerado como ***ginoide*** *(ver figura 4)*, es decir, un robot humanoide con apariencia femenina, fue presentado al publico el 16 de marzo de 2009 por el *AIST* de Tokio. Este robot no fue diseñado para el servicio del hombre así como lo hace ASIMO, sino que fue diseñado puramente para el entretenimiento, especialmente en el campo de la moda. Actualmente este instituto se encuentra en el desarrollo de una versión mejorada, HRP-5C que se habla que tiene un costo de desarrollo de unos 3 millones de dólares.

Su altura es de 1.58 metros y pesa 43 Kg. Incluyendo la batería. Cuenta con 42 motores en total, que sirven para que realice todos sus movimientos de elegancia y coqueteo tratando de imitar a las modelos. En su rostro cuenta con 8 motores que le permiten realizar gestos (permitiéndole mostrar varias emociones) e imitar el movimiento de los labios cuando habla.

Posee una inteligencia artificial que le permite el reconocimiento del habla, es decir, puede percibir cuando una persona le está hablando y puede entender varias órdenes sencillas.

También cuenta con la capacidad de síntesis del habla, que en otras palabras se refiere a que su voz no es producida por grabaciones, sino que posee un software y hardware que convierte texto en sonido, lo que permite que sus diálogos sean fácilmente programable.

2.3.4 Partner robot

Este robot humanoide bastante amigable y de ahí su nombre Partner Robot (ver figura 5) creado por Toyota tiene como principales áreas de aplicación la asistencia y el cuidado de ancianos. Tiene una altura de 120cm y pesa 35 kg. Y tiene la capacidad de utilizar sus manos para muchas tareas. Su gran avance es que puede correr a casi 7 Km/h y se puede mantener en pie si es empujado.

2.3.5 QRIO

QRIO ("**Q**uest for cu**RIO**sity") un robot Humanoide creado por **Sony** (Ver figura 6), mide apenas 60 cm, dispone de una tecnología denominada "Intelligent Servo actuador" que es lo que le permite andar dinámicamente es decir puede variar las r.p.m. y el torque en las articulaciones, y emplea una técnica denominada "Zero Moment Point" para mantener la estabilidad.

Con respecto a las capacidades de QRIO (tales como velocidad, autonomía, entre otras) no se puede nombrar alguna en particular ya que SONY ha estado en continuo desarrollo y cambio de versiones, pero a continuación se muestra una tabla con muchas de sus especificaciones mas generales.

Figura 5: Partener robot está tocando el instrumento

Figura 6: QRIO saludando

3 Diferentes simuladores para robótica

3.1 Introducción

Cuando se realiza un estudio completo para construir un robot humanoide es necesario el uso de la simulación mediante algunos simuladores para poder evaluar el funcionamiento del robot una vez diseñado y así evitar los posibles fallos, colisiones y desajustes con el sistema robótico real montado. El principal motivo para realizar la simulación es la prevención de fallos y posibles daños en los mecanismos del robot y así ahorra el coste del diseño.

En la actualidad existe una gran cantidad de herramientas que permiten realizar simulaciones de robots en un ambiente de tres dimensiones (3D), disponen diversos sensores, actuadores y objetos. Los sensores que se encuentran en estas herramientas son capaces de generar una realimentación entre la interacción del robot con el entorno, el cual debe ser creado también por el usuario. Además, la interacción entre los objetos creados se realiza mediante modelado, teniendo en cuenta la física del sólido rígido.

La mayoría de simuladores utilizan interfaces gráficas para la construcción de los robots y el ambiente en el cual interactúan, y todas admiten diferentes lenguajes para la programación de los controladores, tales como PYTON, Java,C/C++, etc.

Sobre la amplia gama de simuladores disponibles se ha decidido utilizar la aplicación Webots.

3.2 Diferentes Simuladores

A continuación comentamos los simuladores principales en el mercado.

3.2.1 SimRobot

El simulador SimRobot (figura 7) fue desarrollodo por la Universidad de Bremen y el Centro de la investigación alemana de intelegencia artificial. Se está utilizando para la investigación sobre robots autónomos.

Este simulador no está limitado a una clase en especial de robots móviles. El lenguaje utilizado en este simulador está basado en XML, en donde los usuarios son libres de especificar cualquier robot y sus ambientes completamente sin la necesidad de otros lenguajes de programación.

Diferentes partes del cuerpo de un robot, actuadores y múltiples sensores permiten la composición libre de un robot. Para simular la dinámica de cuerpos rígidos, usa Open Dynamics Engine (ODE) y la visualización de las imágenes está basada en OpenGL. [3]

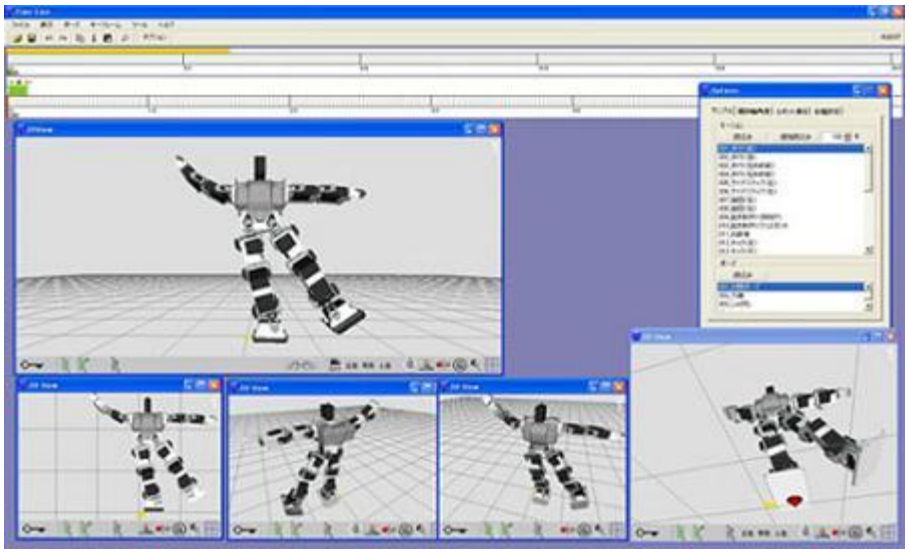

Figura 7 SimRobot

3.2.2 Gazebo

Gazebo (figura 8) fue desarrollado por el Dr. Andrew Howard y su estudiante Nate Koenig en el año 2002 en la Universidad del Sur de California. Sus ideas principales eran buscar y crear un simulador que es capaz de hacer las simulaciones de robot en un entorno interior o exterior.

Gazebo es un simulador 3D, que permite cargar diferentes tipos de robots, sensores y actuadores dentro de un mundo artificial. Gazebo ofrece las primitivas para leer de los sensores e interactuar sobre sus actuadores. También se usa ODE y OpenGL para la simulación de cuerpos rígidos y la visualización de las imágenes.

Sus características principales son: Dispone el modelo de cámara estéreo, genera mapas estéreos de profundidad de la imagen. El interfaz gráfica de usuario escrita en wxPython y se puede usar sin necesidad de instalar el servidor Player. [4]

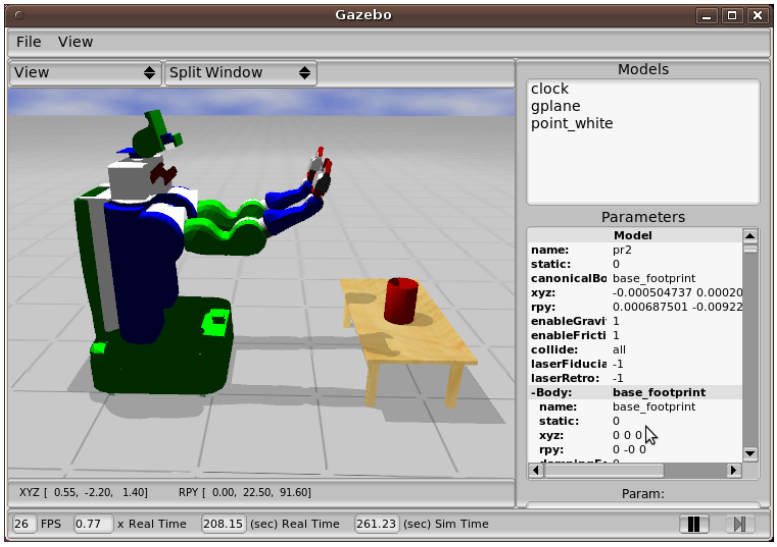

Figura 8 Interfaz de Gacebo

3.2.3 OpenHRP3

OpenHRP3 (Open Architecture Humanoid Robotics Plataform version 3) es una plataforma para simulaciones de robots y desarrollo de software (ver figura 9). Permite a los usuarios inspeccionar el modelo original del robot y el programa de control a través de una simulación dinámica. Además, OpenHRP3 proporciona diversos componentes software de cálculo y bibliotecas que pueden ser utilizadas para desarrollar software relacionados con la robótica.

OpenHRP3 se desarrolla dentro de un importante programa de investigación impulsado por el Ministerio de Economía e Industria de Japón. Forma parte, como proyecto complementario, del llamado "Distributed component type robot simulator", llevado a cabo por "Cooperation of Next Generation Robots". La ingeniería de cálculos dinámicos es desarrollada por "Nakamura Lab, Dept. of Mechano Informatics, University of Tokyo" y la interfaz gráfica es realizada por "General Robotix, Inc". Las otras partes se desarrollan como trabajo entre cooperación de "Humanoid Resarch Group" y "Task-Intelligence Research Group" en los institutos "Intelligent Systems Research Institute" y "National Institute of Advanced Indrustrial Science and Technology (AIST)" .

La descripción de los modelos del robot y del entorno se realiza con el formato VRML. Permite la comprobación de colisiones. También tiene una función para la implementación de diversas leyes de control del robot “Controller”. Existe una función independiente para soportar la dinámica de los mecanismos robóticos y cuenta con una función de visualización de la simulación. Finalmente, dos de los módulos fundamentales son el de planificación del movimiento “MotionPlanner”, que genera trayectorias libres de colisiones y el de generación del patrón de paso “PatternGenerator” que genera movimientos estables para la locomoción del robot basándose en el control del ZMP. Englobando todas esas funcionalidades el OpenHRP3 cuenta con un potente entorno integrado de simulación y constituye un interfaz potentísimo, tanto de desarrollo como de usuario.

Este simulador tiene una inconveniente, ya que su diseño está muy centrado a su propio robot “HRP3”. [5]

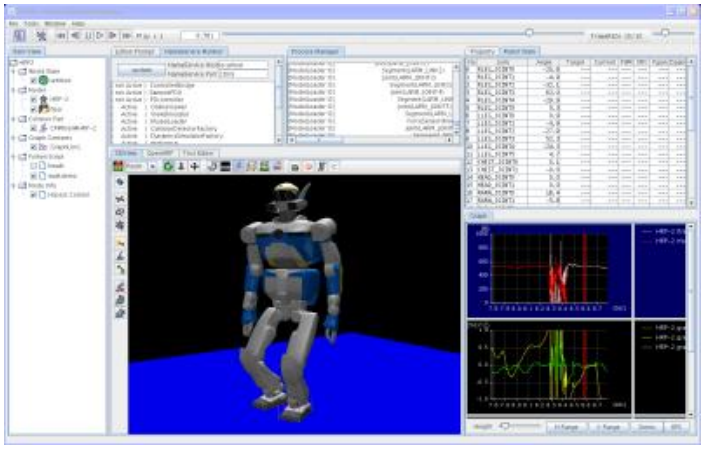

Figura 9 Interfaz de OpenHRP3

3.2.4 Marilou Robotics Studio

Marilou Robotics Studio fue desarrollado por la empresa ANYKODE (ver figura 10). Es un modelador y ambiente de simulación 3D para robots móviles, humanoides, brazos articulados y robots paralelos que funcionan en condiciones verdaderas, respetando las leyes físicas.

En un ambiente realista gráfico, Marilou nos permite crear la jerarquía necesaria para construir y probar ensamblajes de formas simples (cajas, esferas, cilindros, superficies, redes de elevación) y formas complejas, tales como triángulo de malla, con geometrías y formas convexas, posee una simulación en tiempo real o simulación acelerada.

La programación del robot se construye bajo los idiomas C, C++ y C#. en Windows y Linux. No obstante, los creadores dicen que luego viene una versión con el que se podrá utilizar Visual Bacic.Net, y es compatible con Matlab, Java e Intempora RT-Mapas. [6]

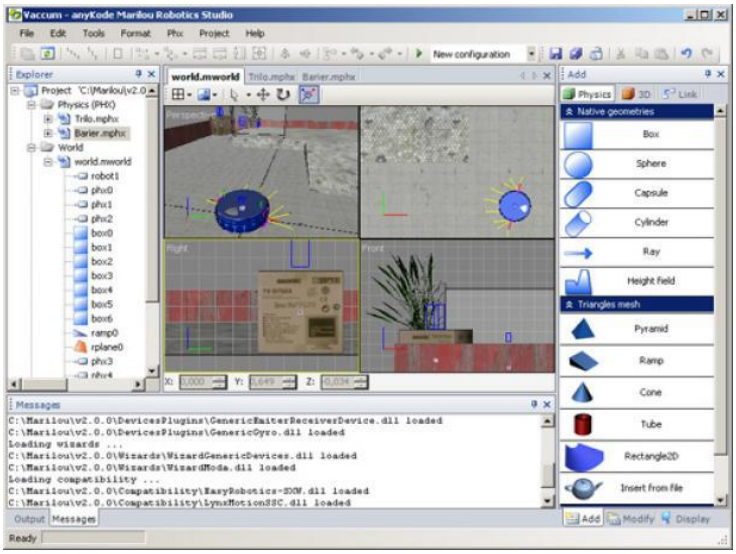

Figura 10: Interfaz de Marilou Robotics Studio

3.2.5 Microsoft Robotics Developer Studio 4

Microsoft Robotics Developer Studio (ver figura 11), fue desarrollado por Microsoft, es una plataforma integrada .NET, se basa en un entorno de desarrollo altamente escalable para distribuida Gamma de aplicaciones y concurrencia de procesos, diseñada para la creación, programación, simulación e implementación de diferentes plataformas Robóticas. Consta de cinco componentes: Concurrencia en Tiempo de

ejecución y Coordinación (CCR), Descentralización de Servicios de Software (DSS), Lenguaje de Programación Visual (VPL) y Entorno de simulación visual (VSE)

La simulación realística está provista por el motor PhysX de AGEIA, el cual posibilita la emulación por software y la aceleración por hardware.

Es una herramienta de programación visual para crear y depurar aplicaciones robóticas. El desarrollador puede interactuar con los robots mediante interfaces basadas en web o en Windows.

La aplicación es una multi-plataforma robótica en donde se permiten varios lenguajes como Visual C#, Visual Basic .NET, JScript, IronPython y lenguajes de terceras partes que se adecuen a la arquitectura basada en servicios. El desarrollador puede acceder fácilmente a los sensores y actuadores de los robots, el simulador proporciona una librería de implementación de concurrencia basada en .NET. La comunicación está basada en mensajes, permitiendo la comunicación entre módulos. [7]

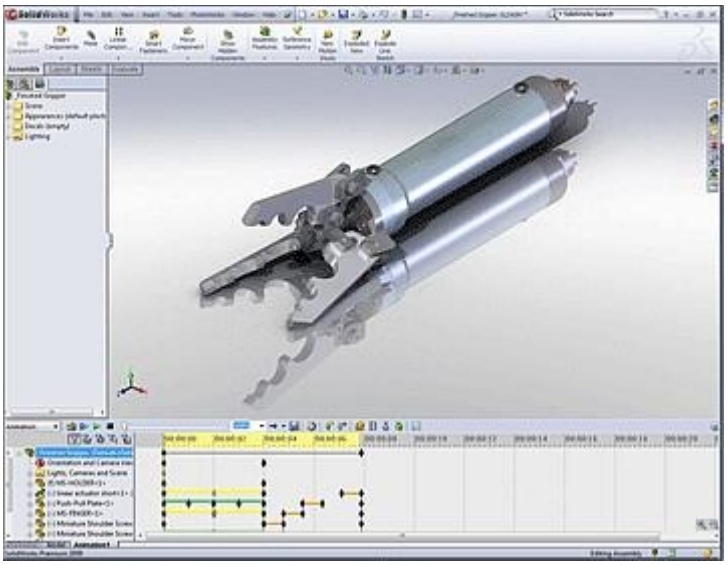

Figura 11: Interfaz de Microsoft Robotics Developer Sudio 4

3.2.6 OpenRAVE

OpenRAVE, fue desarrollado por Rosen Diankov en “the Quality of Life Technology Center” de la Universidad “Carnegie Mellon Robotics Institute”. El proyecto OpenRAVE se inició en 2006 y comenzó como una reescritura completa del RAVE, y luego, se concretó su propia arquitectura y concepto, y comenzó a ser apoyada por muchos investigadores de robótica de todo el mundo.

OpenRAVE está dirigido a aplicaciones del mundo real robot autónomo, e incluye una integración perfecta de 3-D de simulación, visualización, planificación, secuencias de comandos y control (ver figura 12). Una arquitectura de plug-in permite a los usuarios escribir controladores personalizados o ampliar la funcionalidad. Con plugins OpenRAVE, cualquier algoritmo de planificación, de control del robot, o subsistema de detección puede ser distribuido y dinámicamente en tiempo de ejecución, que libera a los desarrolladores de luchar con el código monolítico de las bases.

Los usuarios de OpenRAVE puede concentrarse en el desarrollo de la planificación de secuencias de comandos y los aspectos de un problema sin tener que gestionar de forma explícita los detalles de la cinemática y la dinámica del robot, la detección de colisiones, las actualizaciones de mundo, y el control del robot. La arquitectura OpenRAVE proporciona una interfaz flexible que puede utilizarse en combinación con otros paquetes populares tales como la robótica Player y ROS, porque se centra en la planificación de movimiento autónomo y de alto nivel de secuencias de comandos en lugar de bajo nivel de control y protocolos de mensajes.

OpenRAVE también apoya a una red potente entorno de secuencias de comandos que hace que sea sencillo de controlar y vigilar los robots y el flujo de ejecución durante el cambio de tiempo de ejecución utilizando lenguajes de programación como Python, Octave Y Matlab. Una de las ventajas clave de las arquitecturas abiertas componente es que permiten a la comunidad de investigación de robótica fácilmente compartir y comparar los algoritmos. [8]

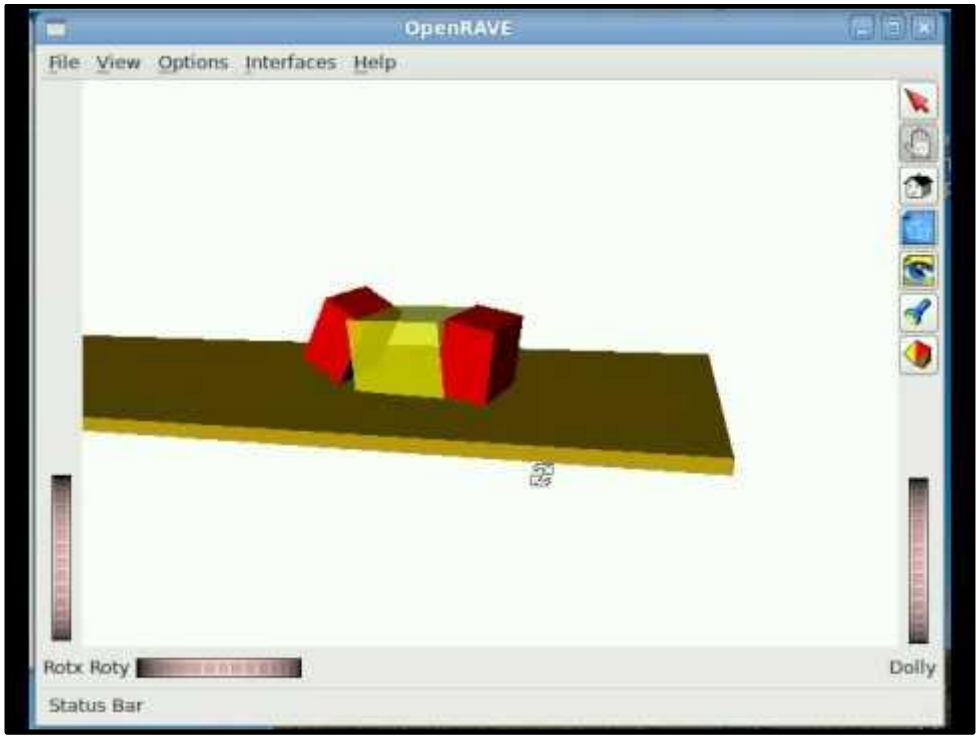

Figura 12: Interfaz de OpenRAVE

4 Simulador Webots

4.1 Descripción general

Webots es un paquete de software profesional para modelar, programar y simular robots móviles. Con este software, el usuario puede diseñar configuraciones complejas de robots, en un entorno virtual en 3D (ver la figura 13). El usuario puede elegir las propiedades de cada objeto, como forma, color, textura, masa, fricción, etc. [10]

Por otra parte, los robots pueden ser equipados con un amplio número de sensores y actuadores, tales como sensores de distancia, ruedas motrices, cámaras, servomotores, sensores de contacto y fuerza, emisores y receptores de señales de radiofrecuencia, etc. Finalmente, el usuario es capaz de programar cada robot individualmente para que exhiba el comportamiento deseado.

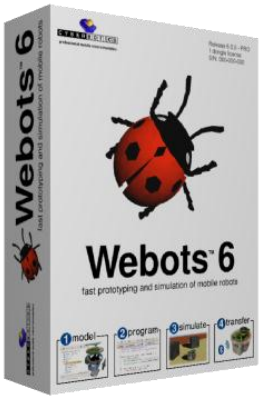

Figura 13: Paquete de software Webots 6

Además, los controladores de los robots se pueden programar en un entorno de desarrollo externo, o en el que hay disponible en Webots. El comportamiento del robot puede ser testeado en

mundos con física realística. Y los programas de los controladores se pueden trasladar a robots físicos reales.

La primera vez que se inicia Webots, hay que establecer cuál será nuestro directorio de trabajo. El asistente creará en este directorio una serie de carpetas, las más importantes de las cuales son “worlds” y “controllers” (ver figura 14), y en ellas se almacenarán los mundos y los controladores creados por el usuario.

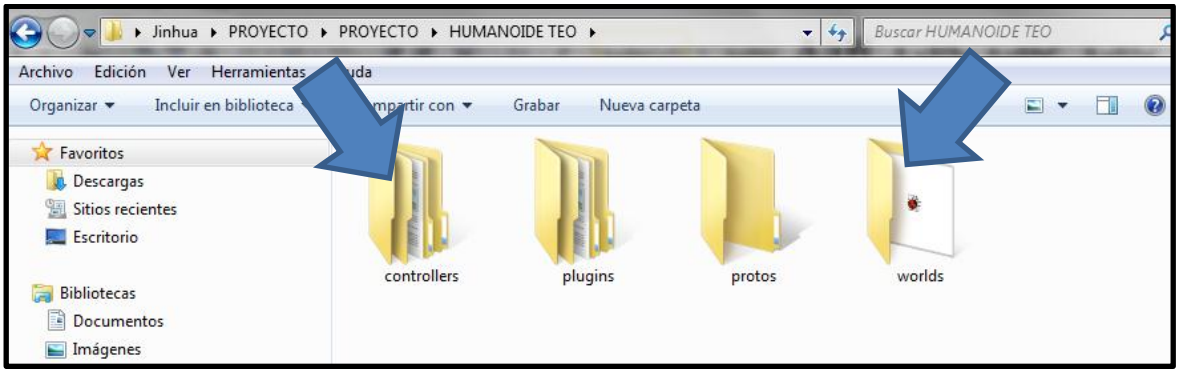

Figura 14: Carpeta generado por el Webots

En la figura 15 se muestra la estructura jerárquica principal de este simulador.

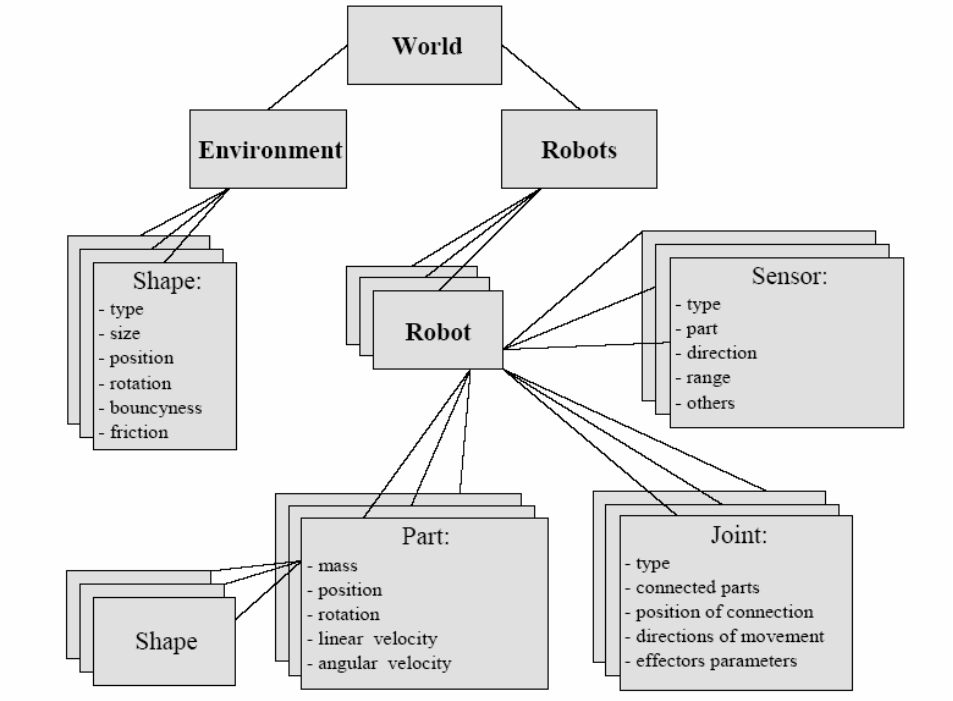

Figura 15: Estructura jerárquica principal de Webots.

4.2 Interfaz

Webots ha sido desarrollado por Cyberbotics Ltd. Tiene versiones tanto para Windows, como para Linux y Mac; para la realización de este PFC hemos utilizado la versión 6.2.4 de Webots que corre bajo el sistema operativo Windows 7. En esta versión la interfaz gráfica se divide en 4 partes (ver figura 16), a continuación, vamos a tratar de explicar la función de cada una de ellas y cómo la hemos utilizado para este proyecto en concreto.

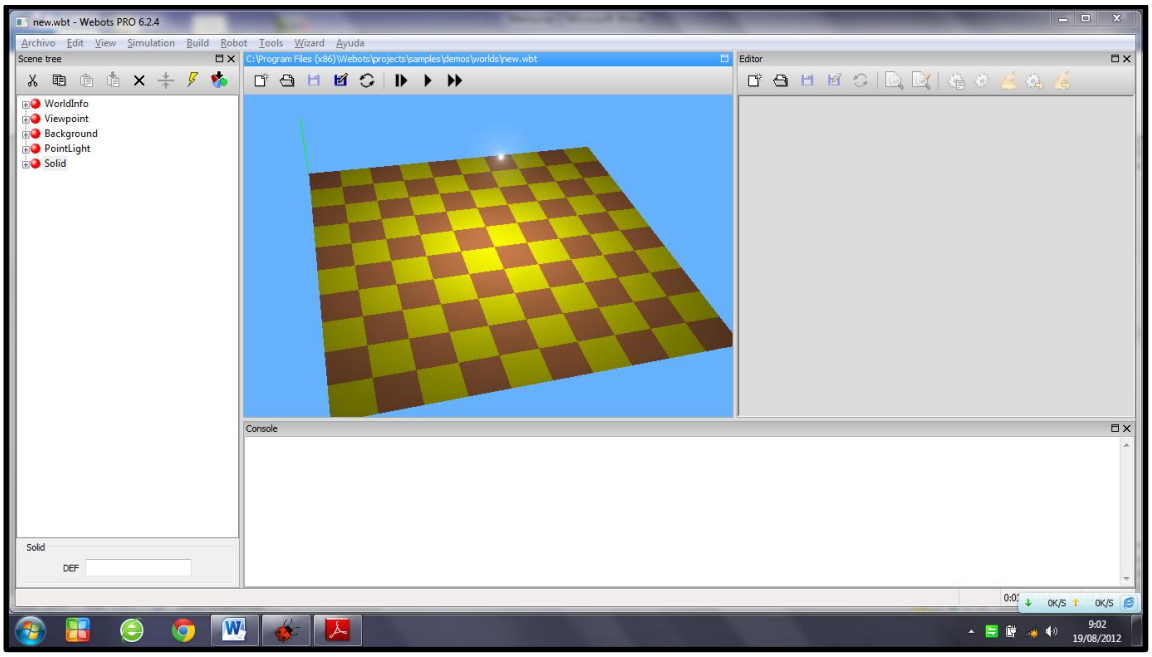

Figura 16: Interfaz de Webots.

4.2.1 Scene Tree

Para poder crear el mundo virtual sobre el que trabajaremos, tenemos una ventana donde se nos muestra el "Scene Tree" o "Árbol de Nodos". Podemos mostrarla u ocultarla haciendo click en *Windows>Scene Tree* en la ventana principal (o Ctrl + T). Se trata de una ventana con la jerarquización de nodos VRML que forman todo el mundo. En esta ventana modificaremos el mundo añadiendo y modificando los nodos que sea necesario. [11]

En la parte superior de esta ventana, tenemos una serie de botones con los que manipular los elementos del árbol de escena (ver figura 17). De izquierda a derecha son: Cortar, copiar, pegar, pegar después del nodo actual, eliminar nodo, resetear a la última configuración guardada, transformar un nodo en otro, insertar un nuevo nodo después del actual, crear un nuevo nodo, exportar, importar y la ayuda.

Figura 17: Menú en la ventana Scene Tree

El árbol de nuestra escena contiene los nodos típicos de todo mundo virtual creado con VRML: ***Worldinfo*** y ***Viewpoint***. Con ellos proporcionamos información de cómo es el mundo y establecemos un punto de vista predeterminado.

4.2.1.1 Nodo Worldinfo

Este nodo proporciona el ambiente de ejecución de los movimientos del robot, y está subdividido en los siguientes campos (ver figura 18):

- ✓ **Campo información del mundo (*info*):** en este campo se puede almacenar información referente al mundo (p.ej. para qué se diseñó, quién lo creó y la fecha del trabajo). Al seleccionar este campo, en la parte inferior se encuentra un cuadro de texto, el cual permite cambiar la información al nodo seleccionado.
- ✓ **Campo gravedad (*gravity*):** en el campo gravedad existen tres valores X, Y y Z respectivamente. Para mantener la simulación lo más realista posible la gravedad de la tierra se mantiene con un valor de -9.81 en el eje de las Y, mientras que en el eje X y Z se mantiene en cero.
- ✓ **Campo de paso de tiempo (*basicTimeStep*):** este campo permite determinar la velocidad en que se ejecutará un movimiento; el valor predeterminado que se usará es de 32 milisegundos.

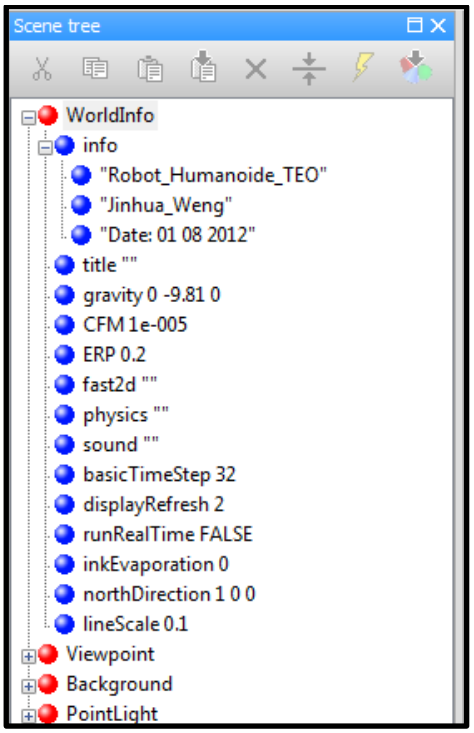

Figura 18: Nodo WordInfo.

4.2.1.2 Nodo Viewpoint

El nodo ***Viewpoint*** permite posicionar a la cámara para interactuar con el mundo de la realidad virtual en un punto determinado (ver figura 19). Para ubicar la cámara es necesario conocer los siguientes campos:

Campo de visión (*fieldOfView*): es el zoom de la cámara, le permite alejar o acercar la cámara del mundo virtual.

Campo orientación (*orientation*): este campo permite hacer una rotación a la cámara en los ejes de X,Y y Z a un águlo Alfa.

Campo posición (*position*): el campo en cuestión permite hacer una traslación al lugar que se le indique.

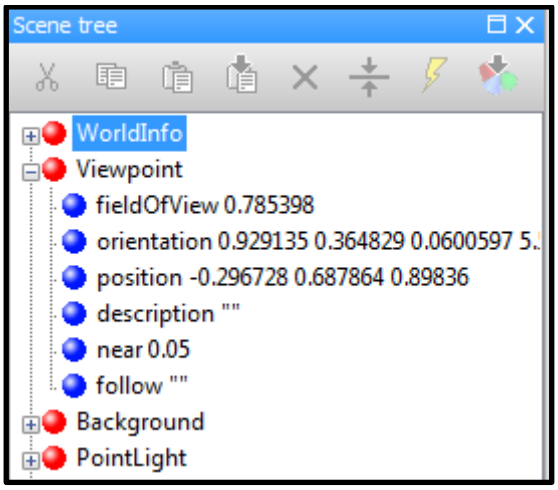

Figura 19: Nodo Viewpoint.

4.2.1.3 Nodo Background.

Este nodo permite establecer un color de fondo gracias al campo que se llama color de cielo (*skycolor*). El color está predeterminado en azul por el sistema (0.4, 0.7, 1). (ver figura 4.2.1.3).

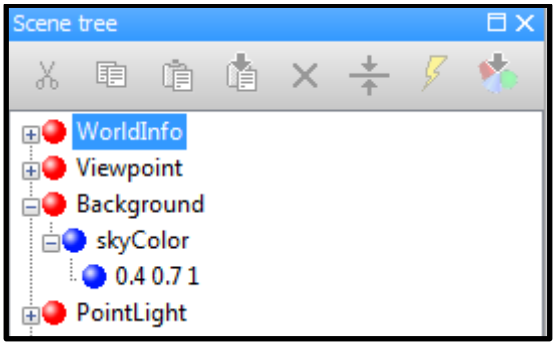

Figura 20: Nodo Background.

4.2.1.4 Nodo Pointlight

El nodo ***Pointlight*** determina la luminosidad, el color, la intensidad, la ubicación de las luces y el radio (ver figura 21). Para darle valores al "foco" es necesario conocer los siguientes campos:

- ✓ **Campo intensidad del ambiente (*ambientIntensity*):** este campo permite controlar la intensidad del ambiente, obtener una luz más brillante o más oscura dependiendo de lo que se desee.
- ✓ **Campo atenuación (*attenuation*):** indica la posición en que la luz se dirige en el campo, se posiciona con las coordenadas (X,Y y Z).
- ✓ **Campo color (*color*):** permite indicar el color de la luz. Por defecto en este proyecto trabajamos con la luz blanca cuyos valores son 1,1,1 en formato RGB.
- ✓ **Campo intensidad (*intensity*):** permite aumentar la intensidad a la imagen,sin embargo, no es recomendable aumentar la intensidad porque los colores se empiezan a perder por la intesidad de la luz.
- ✓ **Campo localización (*location*):** ubica la posición del foco en el campo; con las coordenadas X,Y y Z se determina el origen de la luz.
- ✓ **Campo encendido (*on*):** este campo da la posibilidad de apagar o encender el foco. El programa se establece como *True*, encendido.
- ✓ **Campo de sombra (*castShadows*):** se obtiene sombra de los objetos que están en el campo, por defecto se establece como *False*.

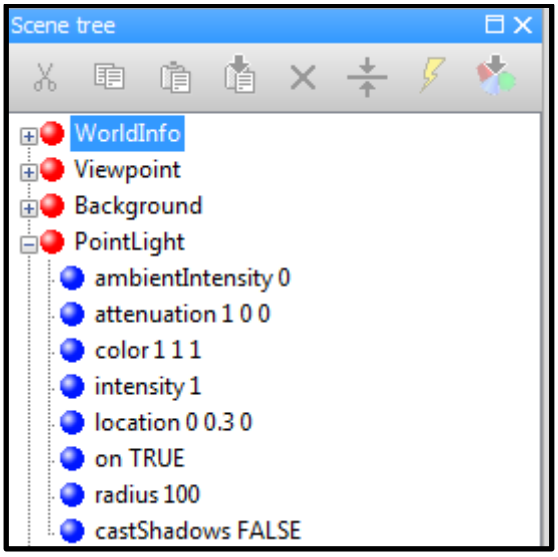

Figura 21: Nodo Pointlight

4.2.2 Text Editor

Una vez creados todos los objetos del mundo y modelado también nuestro robot, debemos programar las acciones que se llevarán a cabo. Lo que nos permite Webots es asociar un programa (controlador) a un robot, de tal forma que podemos recibir información de los sensores que hayamos colocado en nuestro robot y mandar información a los actuadores que también le hayamos incorporado. Estos controladores pueden estar escritos en C, C++ o Java. En este proyecto se ha decidido trabajar en C.

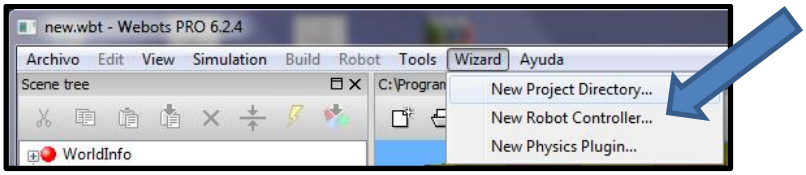

Figura 22: Crear nuevo controlador.

Cuando queremos crear un controlador, debemos hacer click en *Wizard>New robot controller (ver figura 22)*. Entonces elegimos el lenguaje de programación (en este caso es C) desarrollarlo y seguidamente el nombre que le vamos a dar a ese nuevo controlador *"Robot_TEO" (ver figura 23).* Automáticamente Webots creará una carpeta con su nombre dentro de la carpeta controllers del directorio de trabajo.

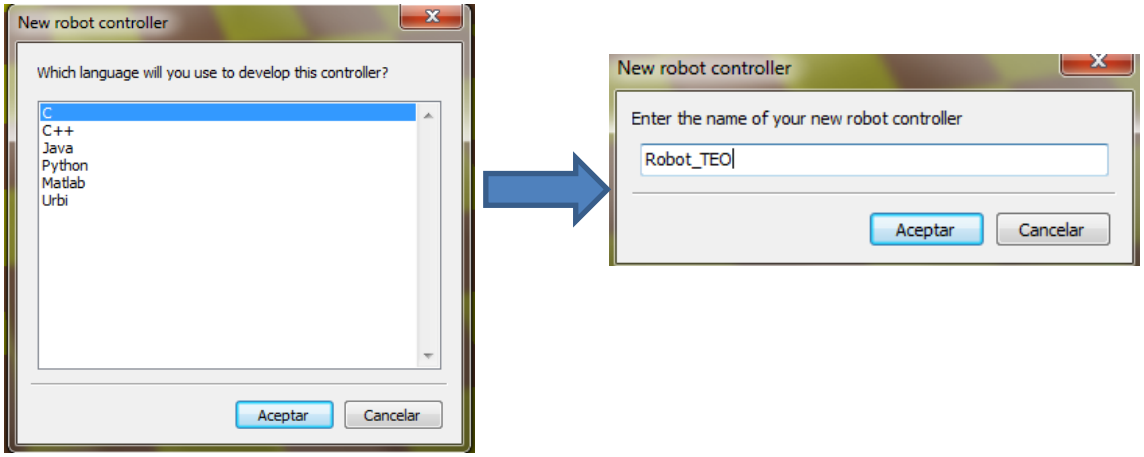

Figura 23: Elegir el lenguaje de programación y asignar el nombre del controlador

De esta forma creamos el controlador y aparecerá una nueva ventana *"Text Editor"*, en esta ventana permite acceder al código del programa para editar y compilar el controlador.

Esta ventana de edición de texto podemos mostrarla desde *Windows>Text editor (ver figura 24)*, en la ventana principal.

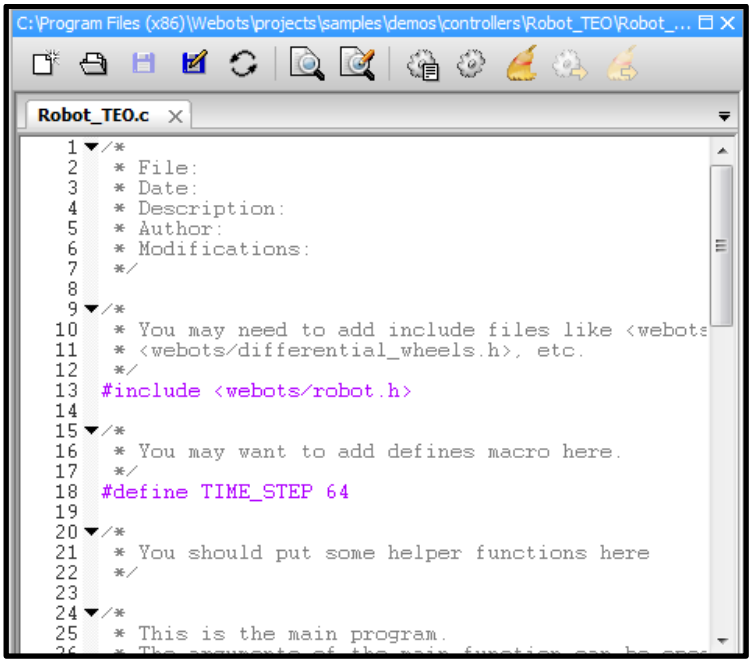

Figura 24: La ventana de Text Editor.

4.2.3 Console

La ventana ***´Console´*** permite desplegar resultados que se están ejecutando en el robot nos va a permitir mostrar todos los datos que necesitemos conocer en el momento de ejecución. para tener una idea de los valores que tiene al interactuar con el medio ambiente y poder utilizar de una mejor manera los actuadores.

Como todas las demás, podemos mostrarla desde *Windows>Log*. Todo lo volcado a esta ventana durante la ejecución se guarda en un fichero *webots.log* en nuestro directorio de trabajo. De esta manera podemos seguir la ejecución y conocer los datos que necesitemos en cualquier momento (ver figura 25).

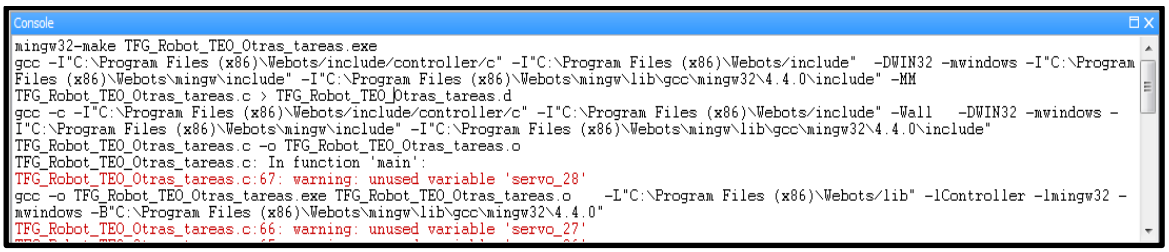

Figura 25: la ventana de Console

4.2.4 3D Window

La ventana ´***3D window***´ muestra los componentes que se establecieron en el *Scene Tree* de una forma visual, y cuando el código finalmente es ejecutado se puede "observar" todo lo que hace el robot en este mundo virtual (ver figura 26).

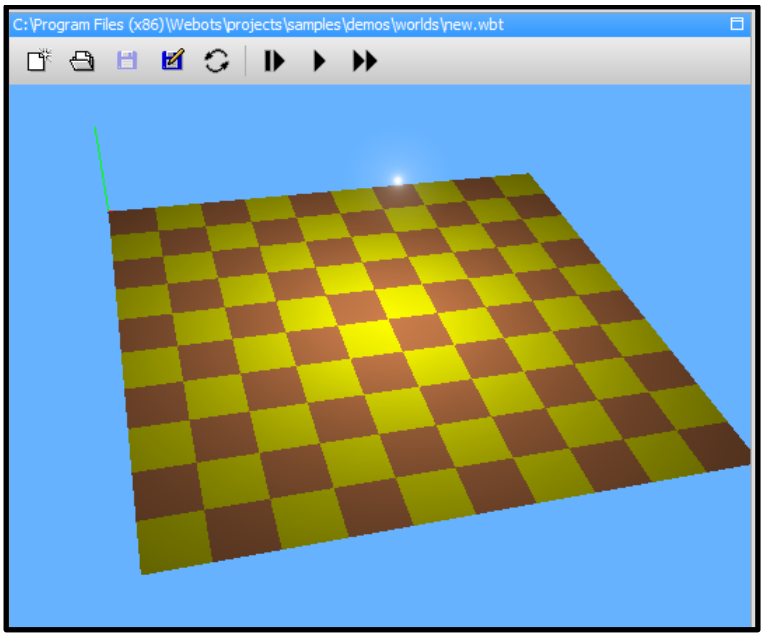

Figura 26: La ventana de 3D

4.3 VRML

En Webots, los objetos se encuentran compuestos por uno o varios nodos. Estos nodos son descritos utilizando el lenguaje VRML, un nodo es la estructura mínima indivisible de un fichero VRML y tiene como misión la de definir las características de un objeto o bien las relaciones entre distintos objetos.

VRML (Virtual Reality Modeling Language) es el lenguaje de programación que se utiliza en Webots para modelar el mundo virtual donde se simulará el comportamiento de los robots. Este lenguaje posibilita la descripción de objetos 3D de formas sólidas situadas y orientadas de determinada forma. Para crear estos mundos de realidad virtual se utilizan ficheros de texto, cuya extensión será siempre .wrl, los cuales pueden ser creados mediante cualquier editor o procesador de textos. Además, existe la posibilidad de utilizar programas de diseño gráfico, los cuales generan automáticamente ficheros en formato VRML, que en nuestro caso es la herramienta Webots.

Los nodos a su vez contienen campos que describen propiedades. Todo campo posee un tipo determinado y no se puede inicializar con valores de otro tipo. De este modo, cada tipo de nodo tiene una serie de valores predeterminados para todos sus campos, de forma que cuando se utilice en una escena sólo se deben indicar aquellos campos que se quieran modificar.

Los campos pueden ser simples, o bien indicando a vectores u otros nodos. En la figura 27 se presenta un ejemplo de programa VRML en Webots, el cual contiene un nodo del tipo *Transform* (transformación) que define el sistema de coordenadas de sus hijos con respecto al sistema de coordenadas del padre. Este nodo contiene tres campos: uno de translación (*translation*) con respecto al nodo padre, uno de rotación (*rotation*) con respecto a sus ejes, y uno de escala (*scale*). Como hijo del nodo *Transform* se tiene el nodo *Shape* (forma), que genera formas tridimensionales en el mundo virtual. Este nodo contiene dos campos: uno de nombre *appearance* (apariencia) el cual define el color y texturas de dicha forma, y otro de nombre *geometry* (geometría), que puede ser una figura geométrica simple (caja, cono, cilindro, etc.) o una extrusión.

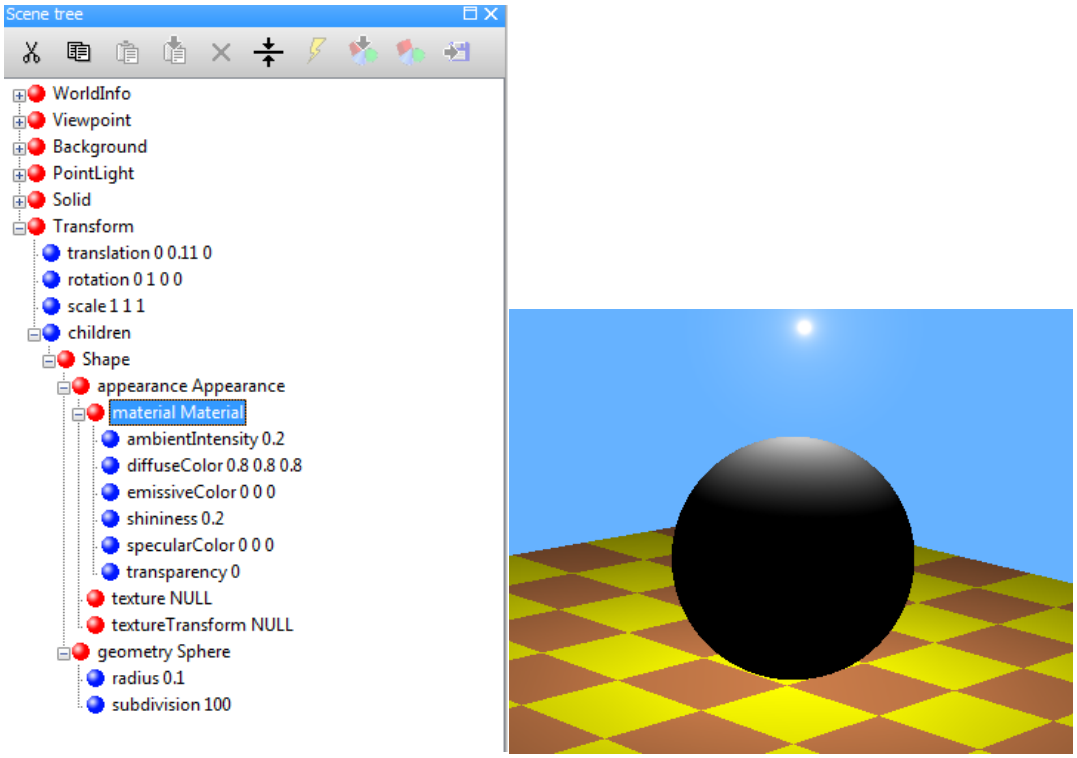

Figura 27: Ejemplo de VRML, creación de una esfera.

4.4 Algunos nodos importantes

La construcción del modelo es necesario conocer los siguientes nodos: nodo Robot, nodo Servo, nodo Transform, nodo Shape, nodo Solid. [11]

4.4.1 Nodo Robot

El nodo ***Robot*** puede ser utilizado como base para la construcción de un robot, por ejemplo, un robot articulado, un robot humanoide, un robot con ruedas, etc. (ver figura 28). Para construir un robot es necesario conocer los siguientes campos:

- ✓ **Controlador (*controller*):** Un controlador es un archivo binario, el cual es usado para controlar a un robot que se ha descrito en un archivo *world*. Los controladores son almacenados en subdirectorios de Webots en el directorio *controllers*. Los controladores pueden ser de diferentes tipos:
  - Archivos binarios de java (.class)
  - Archivos de C (.c y .cpp)
  - Archivos de C++ (.c y .cpp)

  En este proyecto utilizamos lenguaje en C.
- ✓ ***ControllerArgs:*** Es una cadena que contiene los argumentos (separados por espacios) que se pasan a la función main () de programación de C / C + + o el main () de programación en Java.
- ✓ ***Sincronización (synchronization):*** si el valor es TRUE (valor predeterminado), el simulador está sincronizado con el controlador, si el valor es FALSE, el simulador se ejecuta sin controlador. La función wb _robot_get_synchronization() se utiliza para leer el valor de un programa de controlador.
- ✓ ***Batería(battery):*** Este campo debe contener tres valores: la primera corresponde al nivel de energía actual del robot en julios (J), la segunda es la energía máxima en julios, y la tercera es la velocidad de recarga de energía en vatios ([W] = [J] / [s]). El simulador actualiza el valor primero, mientras que los otros dos permanecen constantes. Importante: cuando el valor actual de la energía llega a cero, el robot se para todo el movimiento.
- ✓ ***Consumición de CPU (CPUConsumption):*** El consumo de energía de la CPU (unidad central de procesamiento) del robot en vatios.
- ✓ ***SelfCollision:*** Se establece como TRUE, le permitirá la detección de colisiones en el robot. Esto es útil para los complejos robots articulados para que el controlador no impiden las colisiones internas. Sin embargo, la activación de colisión pueden disminuir la velocidad de simulación. Cuando selfCollision es FALSE (por defecto), Webots no intenta detectar las posibles colisiones internas

en un robot. En este caso, el controlador puede impedir que las diferentes partes del cuerpo de un robot cruzándose entre sí.

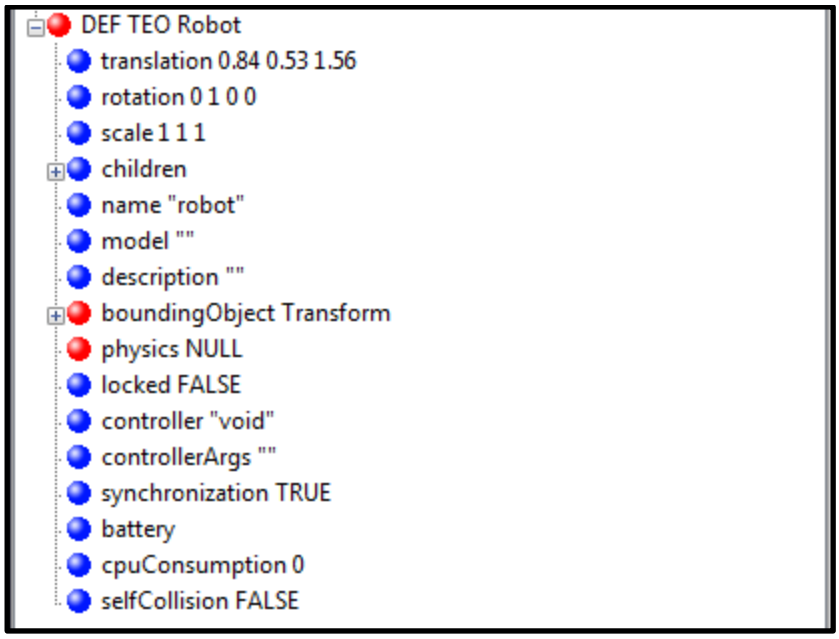

Figura 28: Nodo de Robot en Webots.

4.4.2 Nodo Servo

Un nodo ***servo*** se utiliza para añadir una junta (1 grado de libertad (GDL)) en una simulación mecánica. El movimiento de servo puede ser de tipo "rotacional" o "lineal". Un servo rotacional se utiliza para simular un movimiento de rotación, como un motor eléctrico o una bisagra. Un servo lineal se utiliza para simular un movimiento de deslizamiento, como un motor lineal, un pistón, un hidráulico / cilindro neumático, un resorte o un amortiguador.(ver figura 29). Para construir un robot y sus servos es necesario conocer los siguientes campos:

- ✓ **Tipo *(type):*** Este campo especifica el tipo de movimiento de servo, puede ser "rotacional" (ver figura 30) o "lineal" (ver figura 31).
- ✓ **Máxima velocidad *(maxVelocity):*** En este campo se especifica el límite de la velocidad del servo. La velocidad se puede cambiar en tiempo de ejecución con la función wb_ servo_set_velocity(). El valor debe ser siempre positivo (el valor predeterminado es 10).
- ✓ **Máxima fuerza *(Maxforce):*** En este campo se especifica el límite superior de la fuerza del servo motor. Es una fuerza que está disponible en el motor para llevar a cabo los movimientos requeridos. Esta fuerza se puede cambiar en

tiempo de ejecución con la función wb_servo_set_motor_force().El valor debe ser siempre positivo (el valor predeterminado es 10).

- ✓ **ControlP**: En este campo se especifica el valor inicial del parámetro P. Cuando el valor de P es mayor, se produce un error muy pequeño, y por lo tanto, se obtiene un sistema más sensible. Sin embargo, si el valor de P es demasiado alto, el sistema puede llegar a ser inestable. Si el valor de P es pequeños, se necesitará más pasos/tiempo de simulación para alcanzar la posición de destino, pero el sistema es más estable. Este valor se puede cambiar en tiempo de ejecución con la función wb_ servo_ set_ control_ p().
- ✓ **Aceleración *(aceleration):*** Define la aceleración por defecto de la P-controlador. Si el valor es -1 significa que la aceleración no está limitada por el P_controlador.
- ✓ **Posición *(position):*** Representa la posición actual del servo, en radianes o en metros. Si es un servo rotacional, en radianes. Si es un servo lineal, en metros. Para conocer la posición actual se utiliza la función wb_servo_set_position().
- ✓ ***MinPosition*** y M***axPosition*** especifican la mínima posición y la máxima posición que puede alcanzar el servo motor.

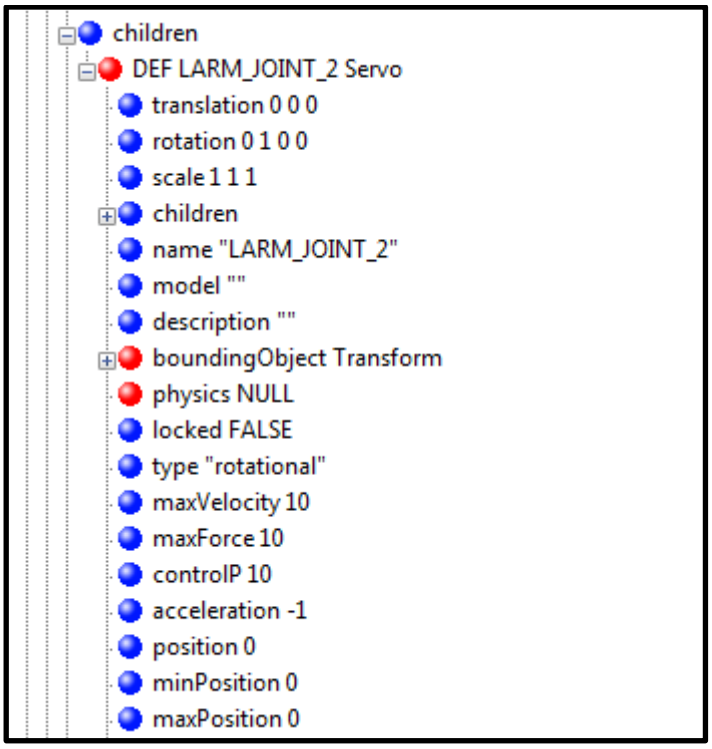

Figura 29: Nodo Servo en Webots.

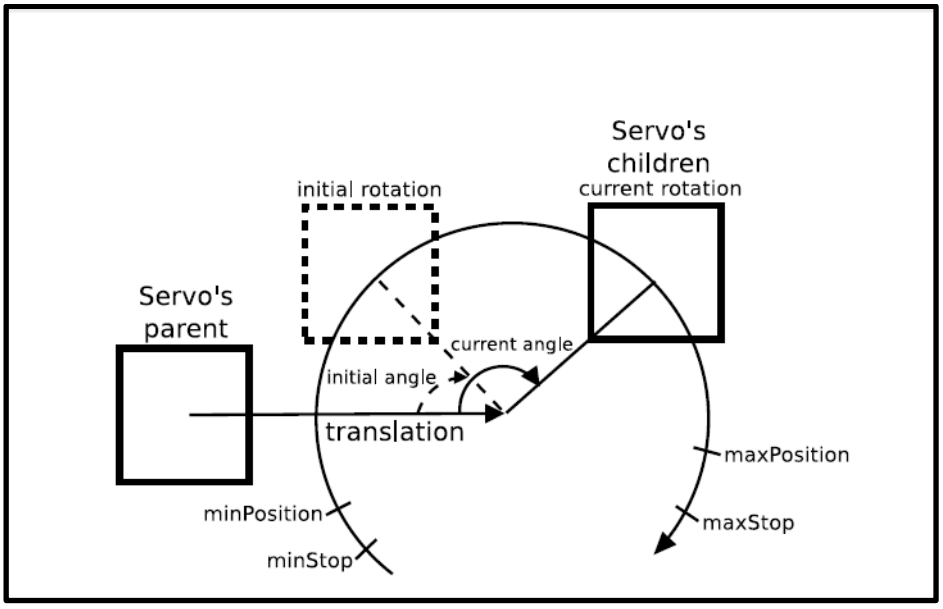

Figura 30: Movimiento de tipo rotacional del servo.

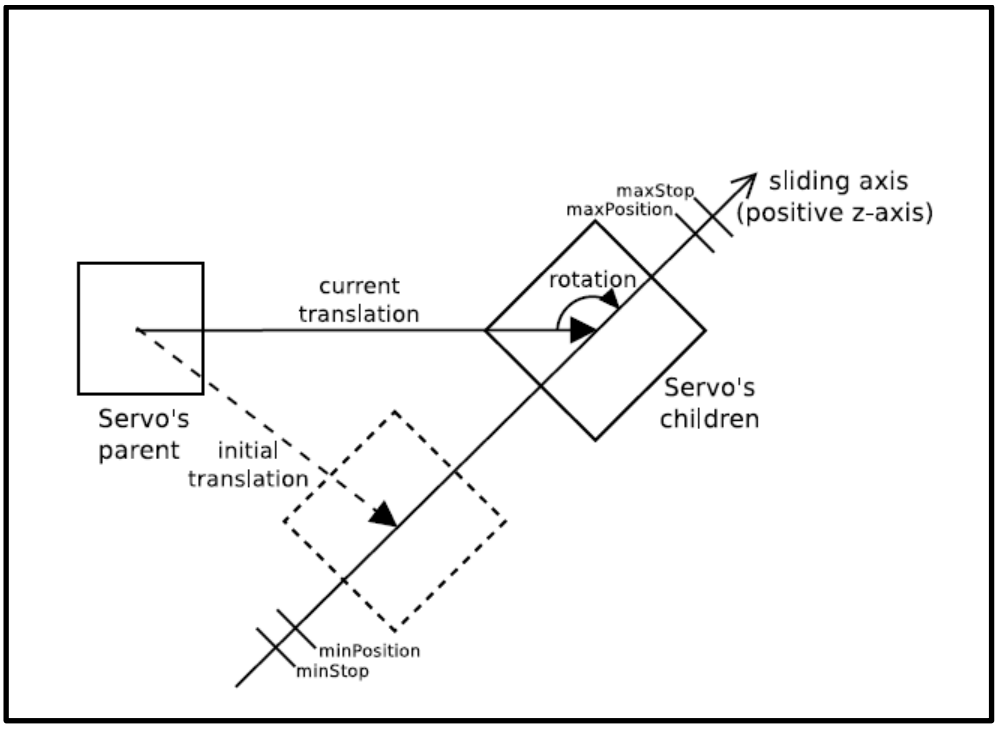

Figura 31: Movimiento de tipo lineal del servo.

4.4.3 Nodo Transform

El nodo ***Transform*** es un nodo de agrupación que define el sistema de coordenadas de nodo hijo con respecto al sistema de coordenadas de nodo padre (ver figura 32).

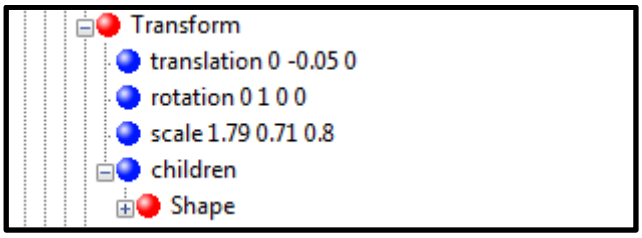

Figura 32: Nodo Transform en Webots.

4.4.4 Nodo Shape

El nodo ***Shape*** tiene dos campos, la apariencia *(appearance)* y la geometría *(geometry),* que se utilizan para crear objetos en el mundo (ver figura 33).

4.4.4.1 Appearance

El campo ***Appearance*** contiene un nodo Apariencia que especifica los atributos visuales (por ejemplo, material y textura) que se aplicará a la geometría.

4.4.4.2 Geometry

El campo de ***Geometry*** contiene un nodo de geometría: Caja, Cono, Cilindro, Extrusión, IndexedFaceSet, IndexedLineSet, plano o una esfera.

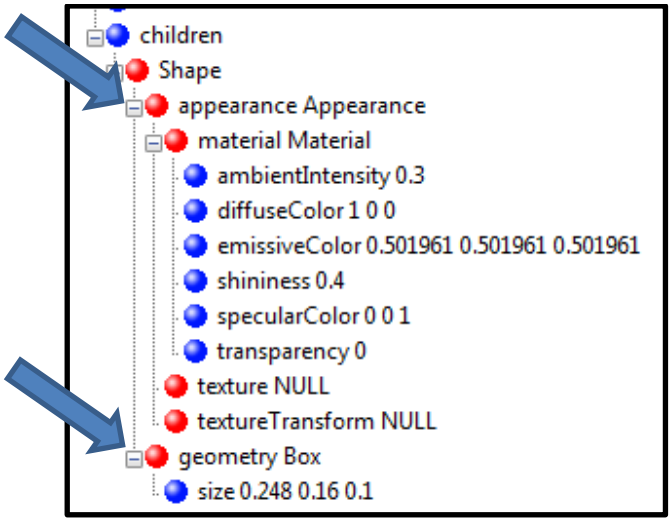

Figura 33: Nodo Shape en Webots.

4.4.5 Nodo Solid

Un nodo ***Solid*** representa un objeto con propiedades físicas, tales como dimensiones, materiales y la masa (ver figura 34). Robot es una subclase de Solid. En la ventana 3D, los nodos sólidos se pueden manipular (arrastrar, levantar, girar, etc.) con el ratón. Es necesario conocer los siguientes campos:

- ✓ **Nombre *(name):*** Es el nombre del sólido. Se utilizado la función wb_robot_get_device () para obtener el nombre.
- ✓ ***BoundingObject:*** Especifica las primitivas geométricas utilizadas para la detección de colisiones. Si el campo ***boundingObject*** es NULL, entonces no se realiza la detección de colisiones y ese objeto puede pasar a través de cualquier otro objeto ("transparente"), por ejemplo, el suelo, los obstáculos y otros robots. Teniendo en cuenta que si el campo ***boundingObject*** es NULL, el campo de la ***Physics*** también debe ser NULL.
- ✓ ***Physics:*** Este campo es un campo opcional, ya que un nodo ***Physics*** que se utiliza para modelar las propiedades físicas de este sólido. Un nodo ***Physics*** deben añadirse cuando conocemos la gravedad, la inercia, fricción y fuerzas, etc.. Si el campo de la física es NULL entonces Webots simula este objeto en modo cinemático. Teniendo en cuenta que si este campo no es NULL, entonces el campo ***boundingObject*** debe ser especificado.
- ✓ **Bloqueado *(locked):*** Si es TRUE, el objeto sólido no se puede mover con el ratón.

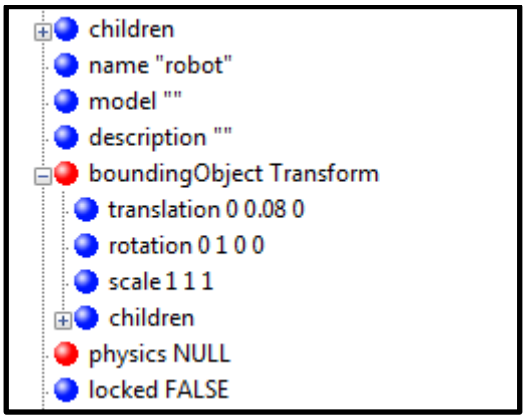

Figura 34: Nodo Solid en Webots.

5 Construcción del Modelo

5.1 Robot Humanoide TEO

Las medidas del TEO son las que se indican en la tabla 1. Estas medidas están calculadas previamente por el departamento de robótica de la universidad Carlos III. Se incluyen todas las medidas originales de cada una de las piezas que forman el robot para conseguir un modelo lo más fiel al robot real.

	L1	L2	L3	L4	L5	L6	L7	L8	L9
[mm]	330	33,22	300	124	146	305	338	337	210

Tabla 1: Las medidas del robot humanoide TEO.

El robot humanoide TEO es un sistema mecánico de 26 grados de libertad (28 si se tienen en cuenta las dos articulaciones en el cuello). Se distribuyen de la siguiente manera por sus extremidades (ver figura 35):

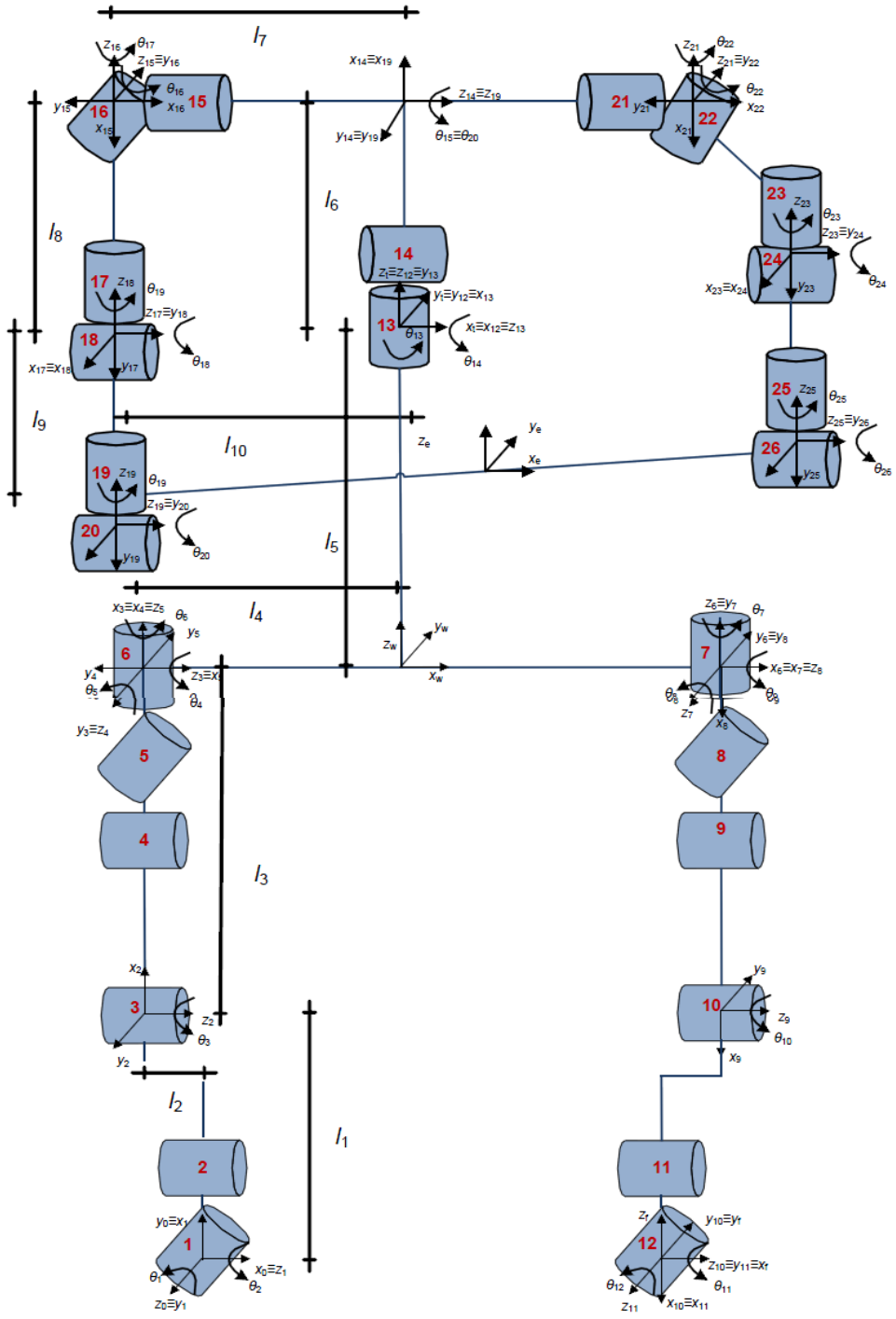

Figura 35: Distribución de GDL del robot TEO

En la tabla 2 se muestra la equivalencia entre los nombres de las articulaciones del robot original y del modelo en el simulador Webots.

Nombres de articulaciones	Número de articulaciones	Nombres de articulaciones del modelo en Webots
Tobillo derecho	1	RLEG_JOINT_6
Tobillo derecho	2	RLEG_JOINT_5
Rodilla derecha	3	RLEG_JOINT_4
Cadera derecha	4	RLEG_JOINT_3
Cadera derecha	5	RLEG_JOINT_2
Cadera derecha	6	RLEG_JOINT_1
Cadera izquierda	7	LLEG_JOINT_1
Cadera izquierda	8	LLEG_JOINT_2
Cadera izquierda	9	LLEG_JOINT_3
Rodilla izquierda	10	LLEG_JOINT_4
Tobillo izquierdo	11	LLEG_JOINT_5
Tobillo izquierdo	12	LLEG_JOINT_6
Tronco	13	BODY_JOINT_1
Tronco	14	BODY_JOINT_2
Espalda derecha	15	RARM_JOINT_1
Espalda derecha	16	RARM_JOINT_2
Codo derecho	17	RARM_JOINT_3
Codo derecho	18	RARM_JOINT_4
Muñeca derecha	19	RARM_JOINT_5
Muñeca derecha	20	RARM_JOINT_6
Espalda izquierda	21	LARM_JOINT_1
Espalda izquierda	22	LARM_JOINT_2
Codo izquierdo	23	LARM_JOINT_3
Codo izquierdo	24	LARM_JOINT_4
Muñeca izquierda	25	LARM_JOINT_5
Muñeca izquierda	26	LARM_JOINT_6
Cuello	27	HEAD_JOINT_1
Cuello	28	HEAD_JOINT_2

Tabla 2: Equivalencia de articulaciones.

En la figura 36 se muestra el modelo del robot en el simulador Webots.

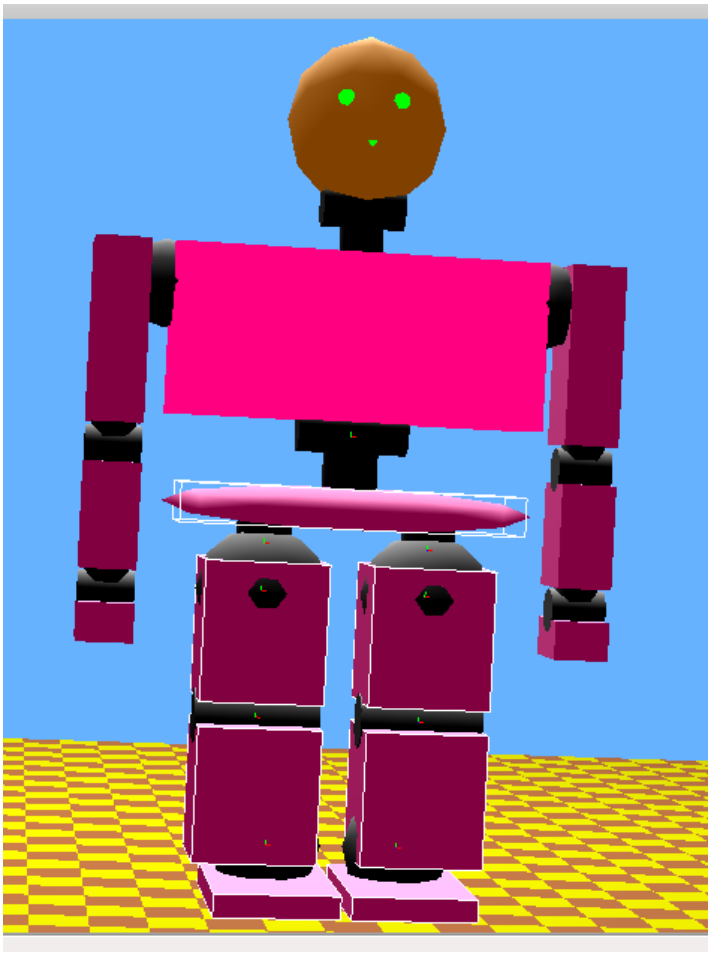

Figura 36: Modelo del robot TEO en Webots.

En la figura 37 se muestra el robot TEO en el laboratorio.

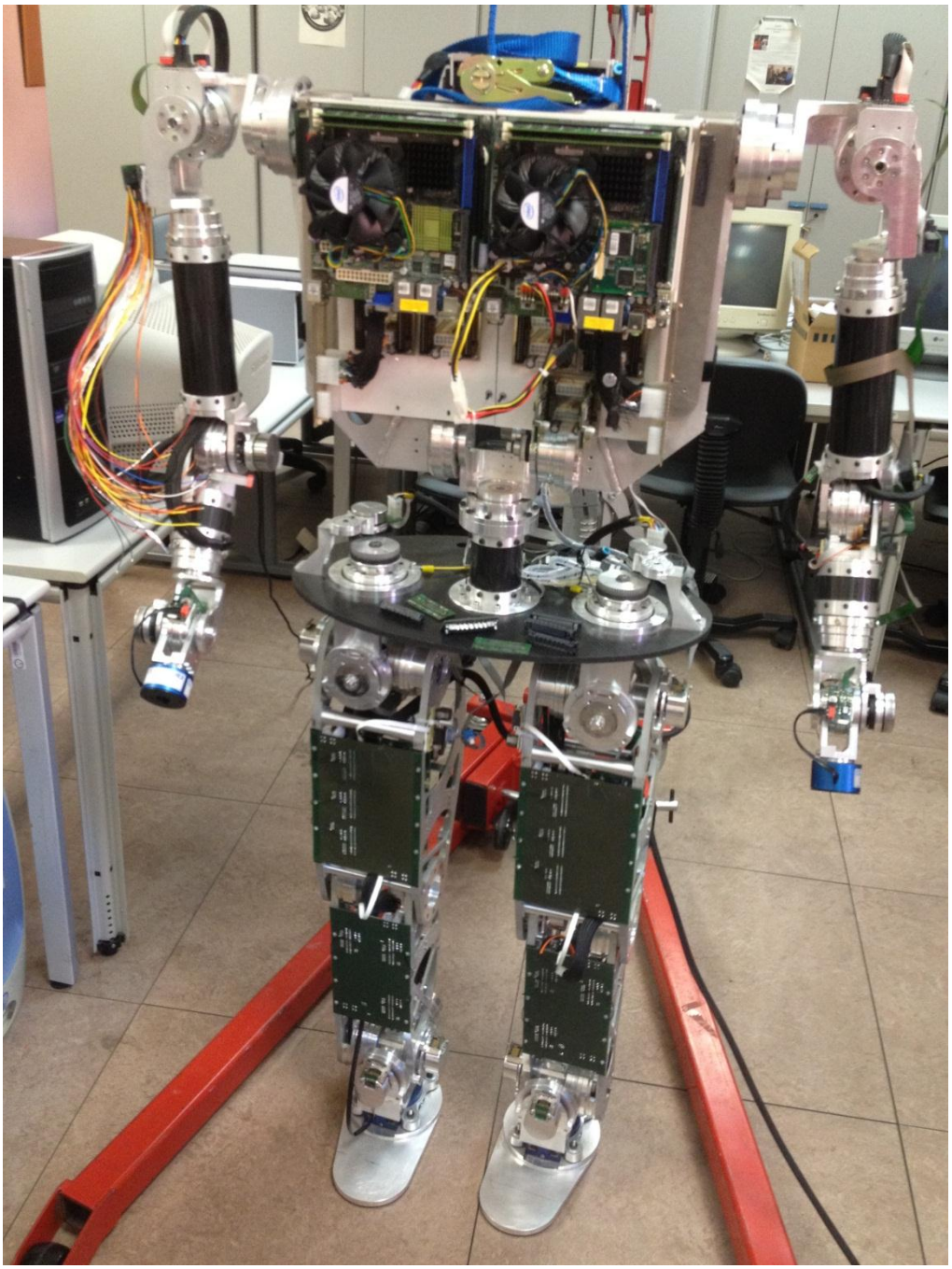

Figura 37: Robot TEO en el laboratorio

5.2 Estructura de nodos del Robot TEO

La estructura jerárquica de un mundo virtual en VRML está compuesta por un árbol de nodos (*Scene Graph*), donde cada círculo representa un Nodo (*Node*) que posee cierta funcionalidad y los nodos padres agrupan a otros nodos hijos.

En la figura 38 se muestra la estructura jerárquica de un mundo virtual, *"Nodo 1"* es el padre de *"Nodo 2"* y *"Nodo 3",* sin embargo, *"Nodo 2"* es padre de *"Nodo 4", "Nodo 5"* y *"Nodo 6",* y *"Nodo 7"* y *"Nodo 8"* son hijos de *"Nodo 5".*

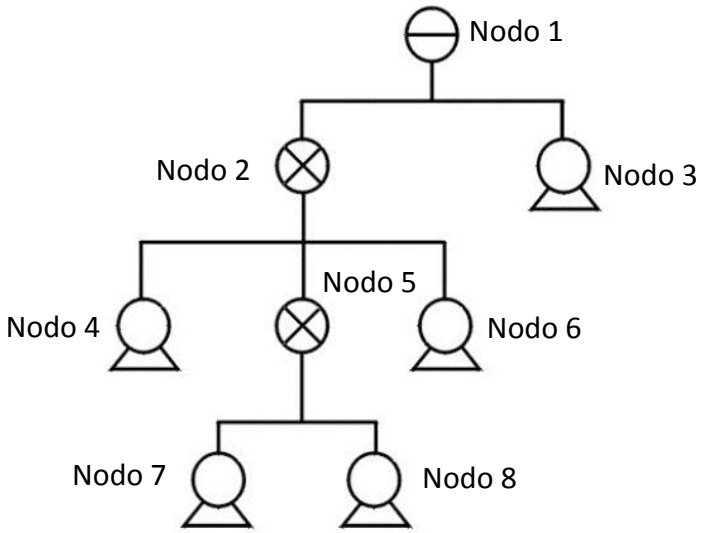

Figura 38: La estructura jerárquica de un mundo virtual en VRML.

Nuestro robot TEO representa 2 nodos hijos (ver figura 39), el tronco (BODY_JOINT_[1]) y la cadera. La cadera tiene tres nodos: su forma del eslabón (esfera), la articulación "LLEG_JOINT_[1]" y la articulación "RLEG_JOINT_[1]". A continuación se muestra la estructura de nodos principales de nuestro robot TEO.

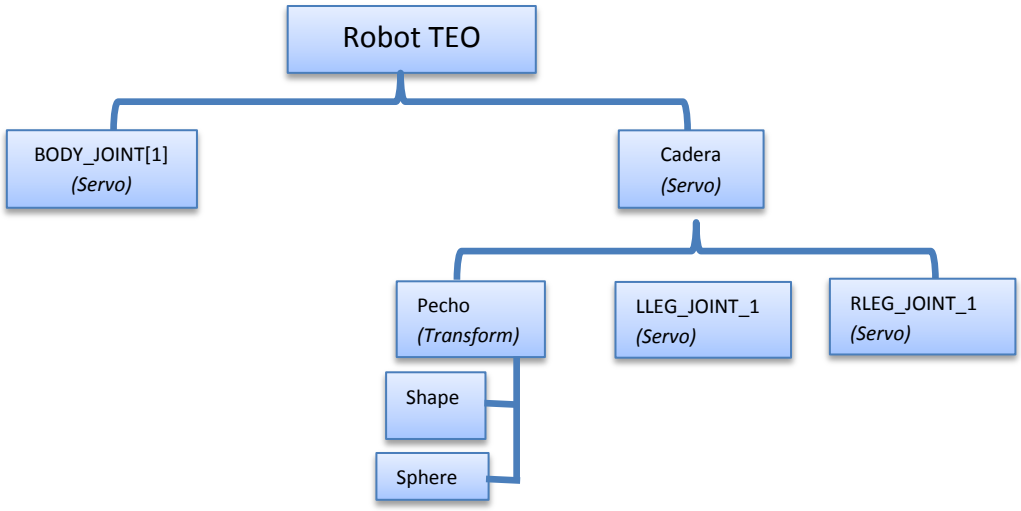

Figura 39: Estructura jerárquica de los nodos principales del robot TEO.

En la figura 40 se muestra la estructura de los nodos de tronco de robot TEO. La articulación "BODY_JOINT_[1]" contiene dos nodos, uno es la forma del eslabón de esta articulación (cilindro), y otro es la articulación "BODY_JOINT_[2]" .

La articulación "BODY_JOINT_[2]" tiene 5 nodos: su forma del eslabón (cilindro), la forma del eslabón del pecho (caja), la articulación "HEAD_JOINT_[1]", la articulación "LARM_JOINT_[1]", la articulación "RARM_JOINT_[1]".

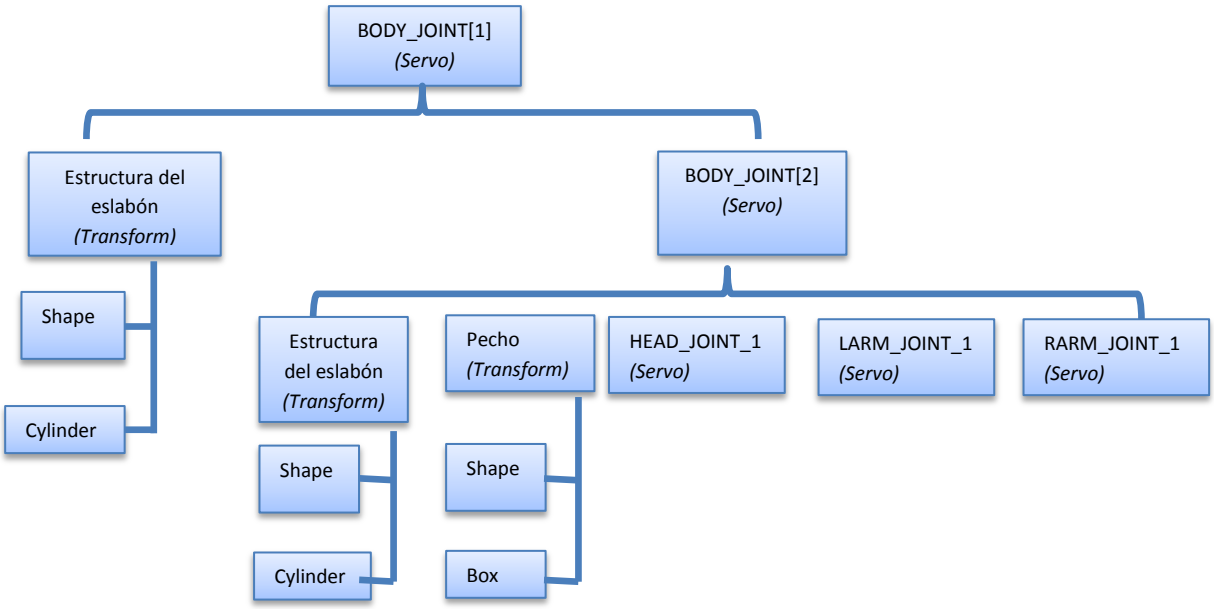

Figura 40: Estructura jerárquica de los nodos de tronco de robot TEO

En la figura 41 se muestra la estructura de los nodos de cabeza de robot TEO. La articulación "HEAD_JOINT_[1]" tiene tres nodos: su forma del eslabón de esta articulación (cilindro), y otro es la articulación "HEAD_JOINT_[2]" .

La articulación "HEAD_JOINT_[2]" tiene 2 nodos: su forma del eslabón (cilindro), la forma del eslabón de la cabeza (esfera).

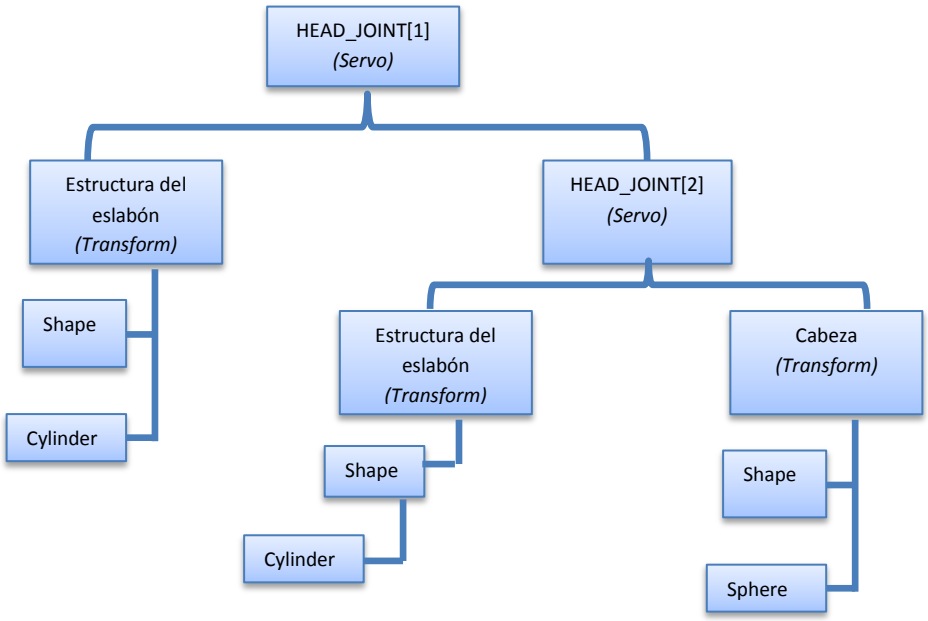

Figura 41: Estructura jerárquica de los nodos de cabeza de robot TEO

En la figura 42 se muestra la estructura de los nodos de brazo derecho de robot TEO. La articulación "RARM_JOINT_[1]" comparte el mismo eslacón con la articulación "RARM_JOINT_[2]", debido a la forma de esfera tiene dos GDL.

La articulación "RARM_JOINT_[3]" tiene tres nodos: su forma del eslabón (cilindro), el eslabón del brazo superior (caja) y la articulación "RARM_JOINT_[4]".

La articulación "RARM_JOINT_[4]" tiene tres nodos: su forma del eslabón (cilindro), el eslabón del brazo inferior (caja) y la articulación "RARM_JOINT_[5]".

La articulación "RARM_JOINT_[5]" tiene dos nodos: su forma del eslabón (cilindro) y la articulación "RARM_JOINT_[6]".

La articulación "RARM_JOINT_[6]" tiene dos nodos: su forma del eslabón (cilindro) y el eslabón de la mano (caja).

Debido a la estructura de los nodos de sus dos brazos son iguales, bastaría explicar con lafigura 42 para el brazo izquierdo.

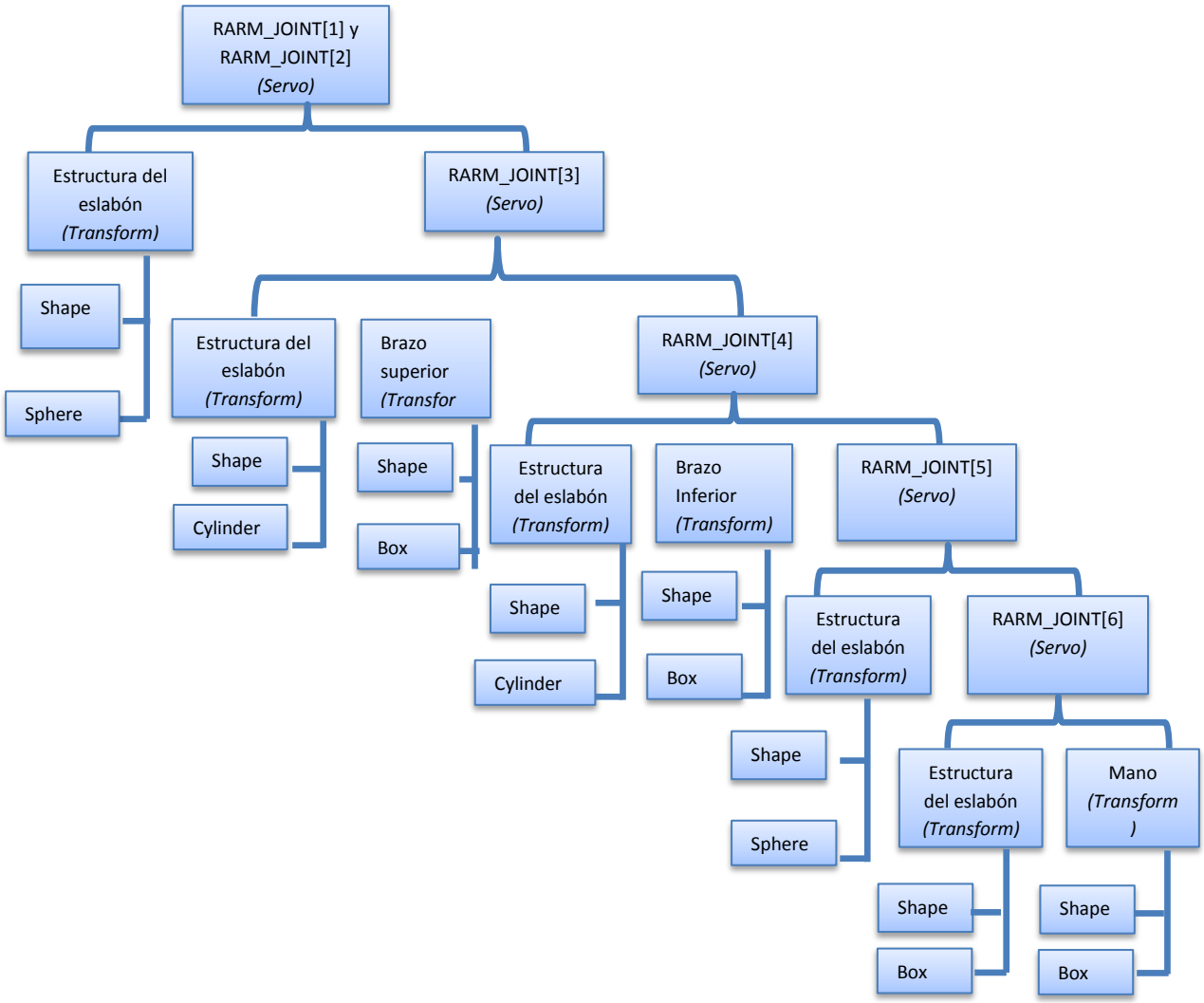

Figura 42: Estructura jerárquica de los nodos de brazo derecho del robot TEO

En la figura 43 se muestra la estructura de los nodos de pierna derecha de robot TEO. La articulación “RLEG_JOINT_[1]” tiene dos nodos: su forma del eslabón (cilindro) y la articulación “RLEG_JOINT_[2]” y “RLEG_JOINT_[3]”. La articulación “RLEG_JOINT_[2]” comparte el mismo eslabón con la articulación “RLEG_JOINT_[3]”, debido a la forma de esfera tiene dos GDL.

La articulación “RLEG_JOINT_[3]” tiene tres nodos: su forma del eslabón (cilindro), el eslabón de la pierna superior (caja) y la articulación “RLEG_JOINT_[4]”.

La articulación “RLEG_JOINT_[4]” tiene tres nodos: su forma del eslabón (cilindro), el eslabón de la pierna inferior (caja) y la articulación “RLEG_JOINT_[5]” y “RLEG_JOINT_[6]”. La articulación “RLEG_JOINT_[5]” comparte el mismo eslabón con la articulación “RLEG_JOINT_[6]”, debido a la forma de esfera tiene dos GDL.

La articulación “RLEG_JOINT_[6]” tiene dos nodos: su forma del eslabón (cilindro) y el eslabón del pie (caja).

Debido a la estructura de los nodos de sus dos piernas son iguales, bastaría explicar con lafigura 43 para la pierna izquierda.

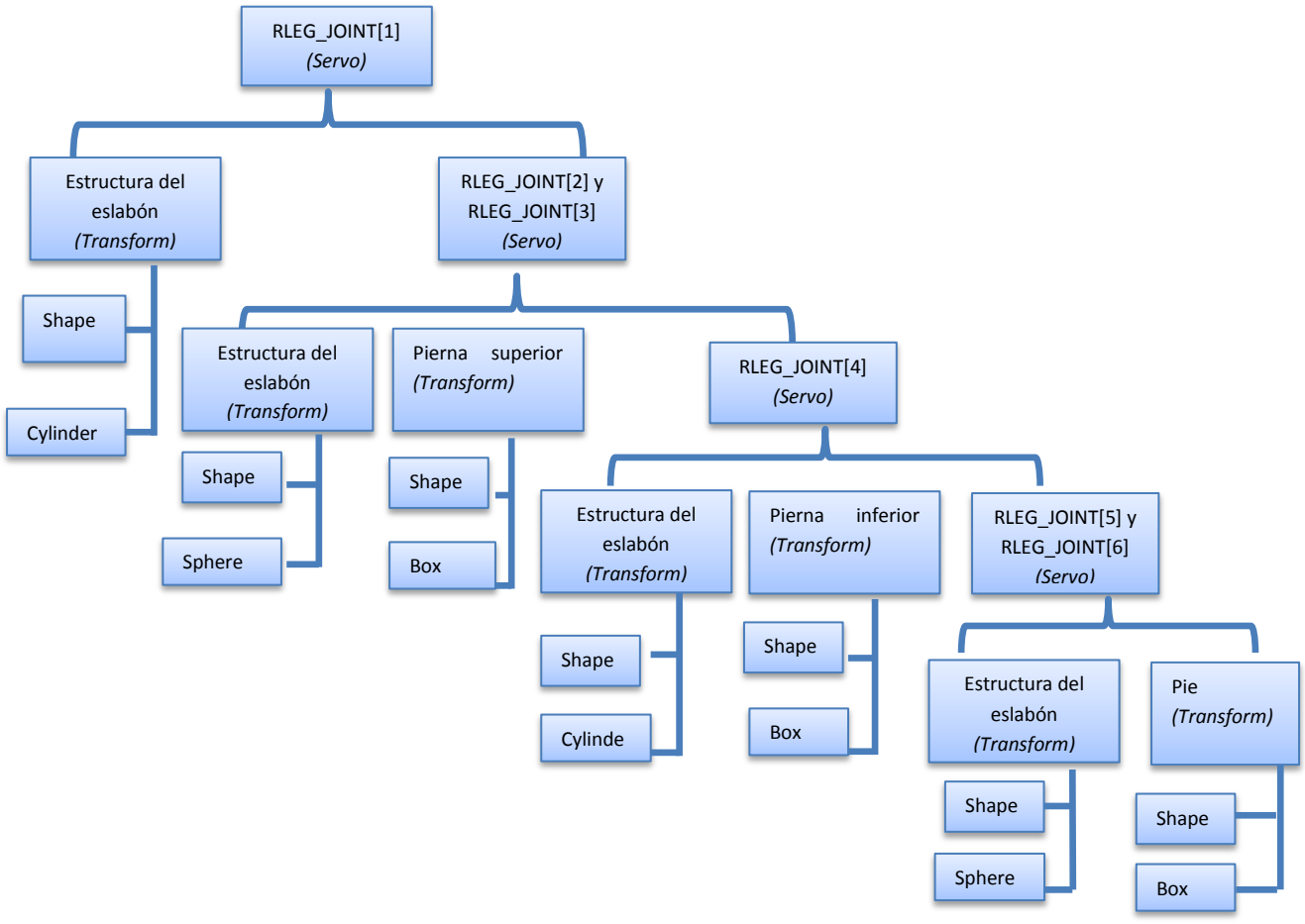

Figura 43: Estructura jerárquica de los nodos de pierna derecha de robot TEO

5.3 Grados de libertad del Robot TEO

5.3.1 Grados de libertad de la pierna

Cada pierna dispone de 6 GDL (grados de libertad) distribuidos entre el tobillo, la rodilla y la cadera. Este número es ideal para un robot humanoide, puesto que el robot pueda desplazarse en línea recta y poder girar libremente de forma humana. En la siguente foto (figura 44) se muestran las dos piernas del robot TEO.

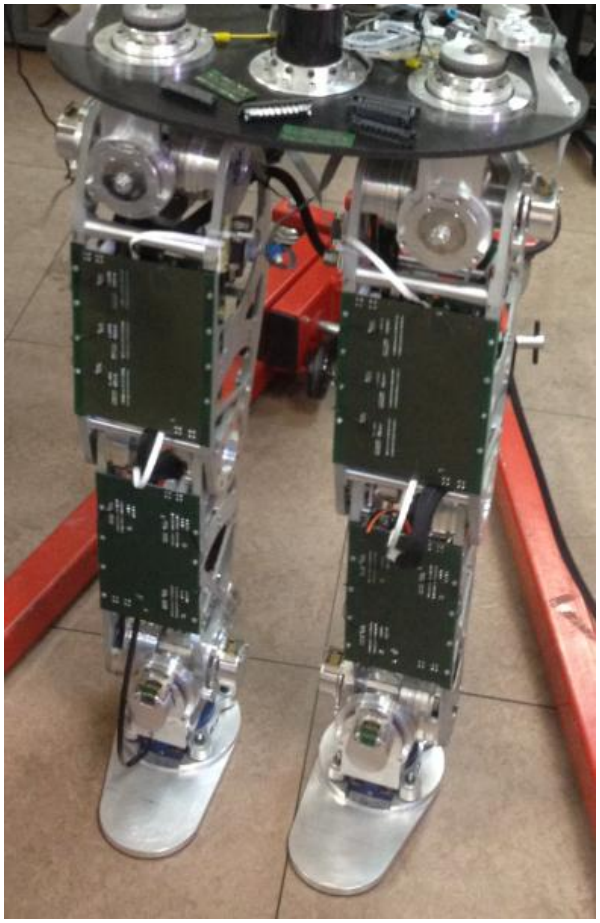

Figura 44: foto de la cadera de robot TEO en el laboratorio.

5.3.1.1 Cadera

La cadera es la que posee un mayor número de grados de libertad. Estos son tres y todos son rotatorios, sobre el eje X, Y e Z, tal como se muestra en la Figura 45.

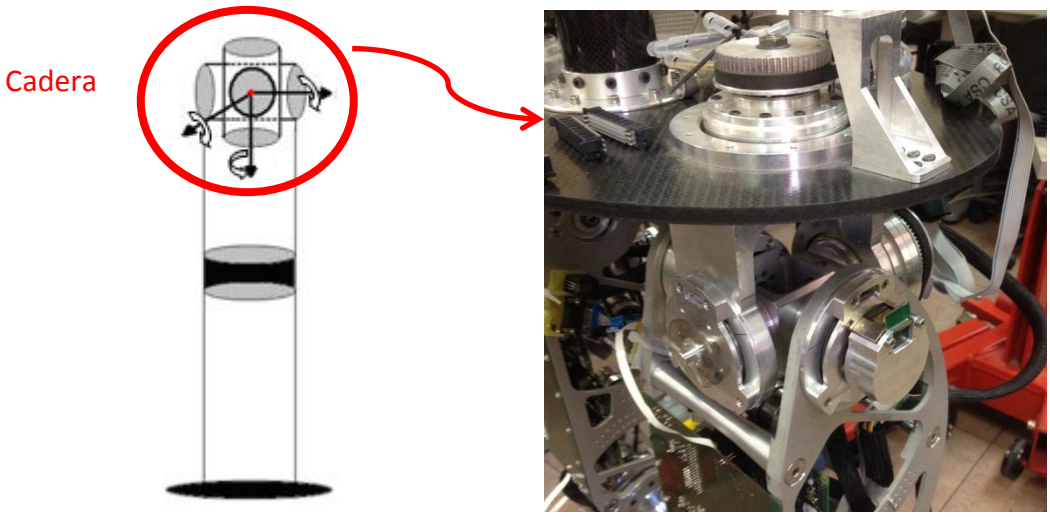

Figura 45: Grados de libertad de la cadera.

La articulación cuyo eje es el X se utiliza para el desplazamiento de la pierna, hacia delante y hacia atrás, la articulación en el eje Y es para que toda la pierna pueda rotar sobre su propio eje y la articulación en el eje Z es para poder levantar la pierna, lo que permitirá al robot no sólo desplazarse en línea recta sino también poder girar.

5.3.1.2 Rodilla

La rodilla es la parte mecánica más fácil de implementar que las demás, solo posee un grado de libertad cuyo eje sería en el eje X, tal como se muestra en la Figura 46.

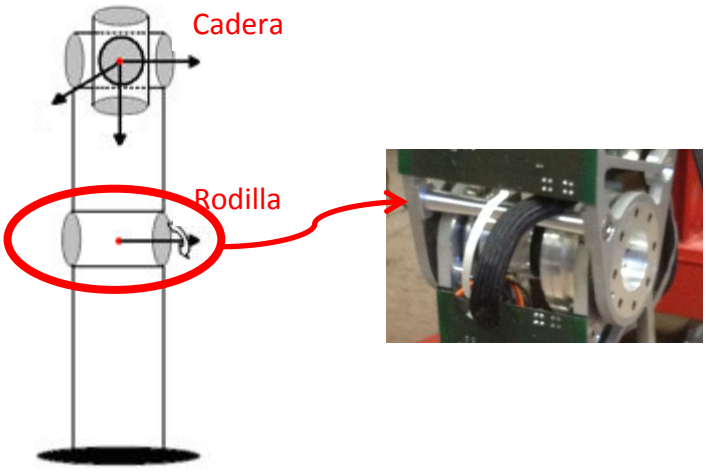

Figura 46: Grados de libertad de la rodilla.

5.3.1.3 Tobillo

El tobillo cuenta con dos grados de libertad en el eje X y Z (ver la figura 47), el grado de libertad del eje X es más importante que el Z, puesto que este grado de libertad es el que permite balancear el peso del robot hacia el pie de apoyo, para que el centro de masa del robot se encuentre en todo momento dentro del área de soporte del pie sobre el terreno (balanceo estático).

El grado de libertad en el eje Z es útil para contrarrestar el peso, ya sea hacia delante o hacia atrás cuando uno o los dos pies se encuentren sobre el suelo.

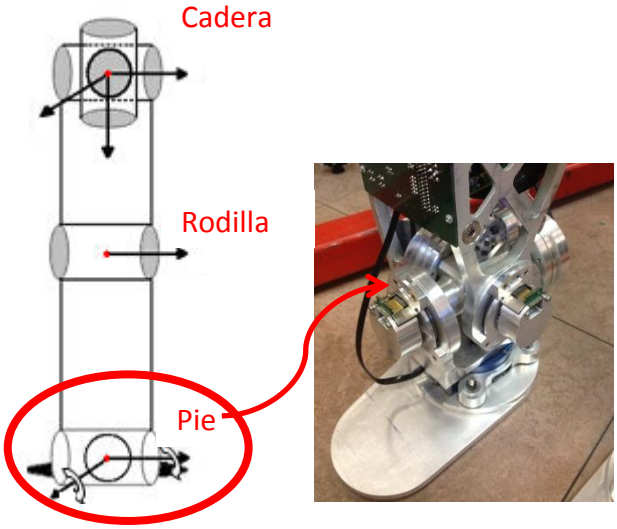

Figura 47: Grados de libertad del tobillo.

En la Figura 47 se puede evidenciar que las piernas del robot que se diseñó constan de 6 grados de libertad cada una, lo que hace un total de 12 grados de libertad, que es un número de grados suficientes para que el robot pueda desplazarse en línea recta e imitar la forma de caminar humana.

5.3.2 Grados de libertad del brazo

Cada uno de los brazos dispone de 6 GDL distribuidos entre el hombro, el codo y la muñeca. Le permite al movimiento del brazo en un espacio tridimensional, es decir, la capacidad de moverse hacia delante/atrás, arriba/abajo, izquierda/derecha (translación en tres ejes perpendiculares), combinados con la rotación sobre tres ejes perpendiculares. El movimiento a lo largo de cada uno de los ejes es independiente de los otros, y cada uno es independiente de la rotación sobre cualquiera de los ejes. En la siguente foto (figura 48) se muestran el brazo izquierdo del robot TEO.

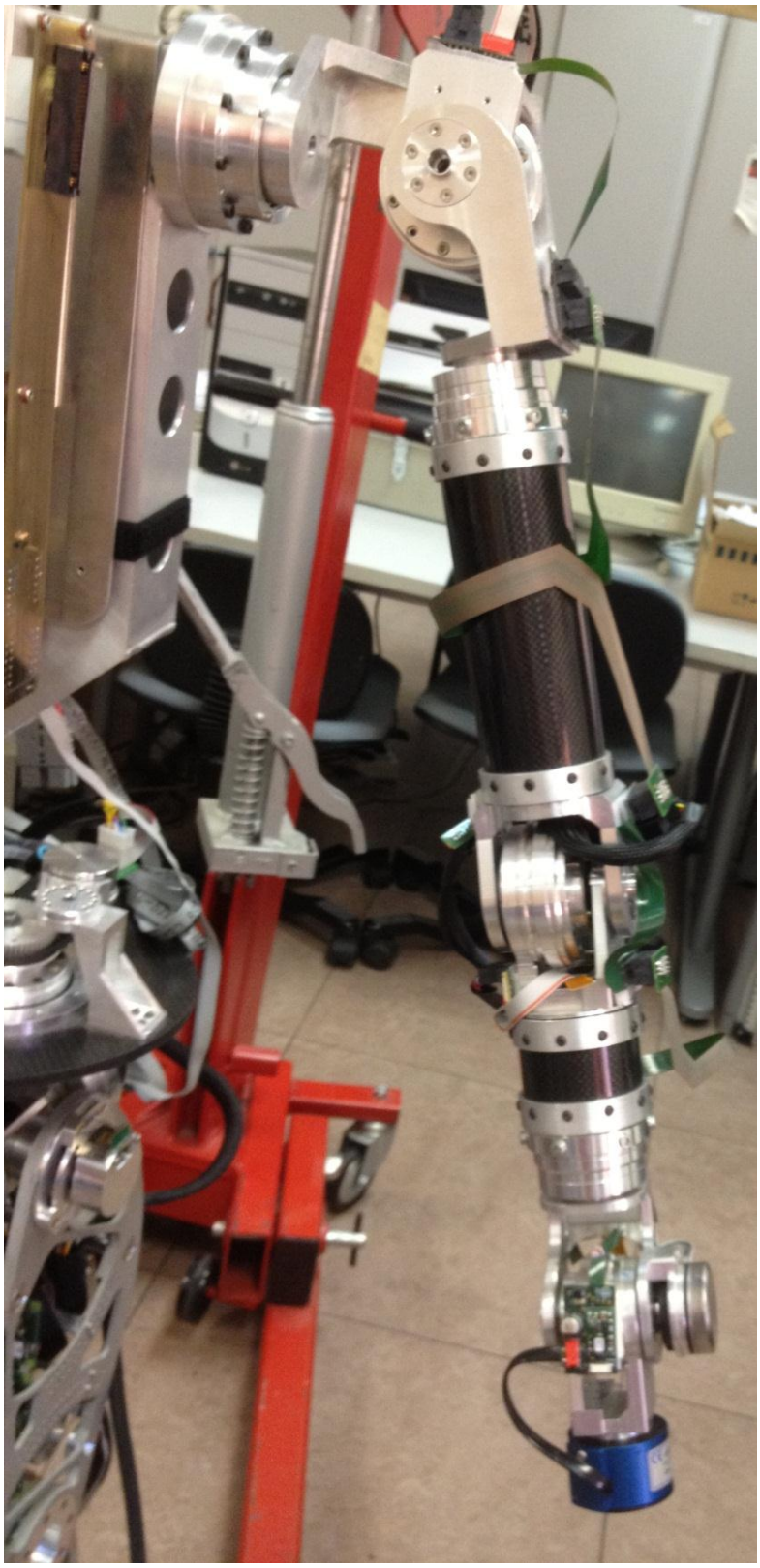

Figura 48: Foto del brazo del robotTEO en el laboratorio.

5.3.2.1 Hombro

El hombro tiene dos grados de libertad y todos son rotatorios, sobre el eje X y Z, tal como se muestra en la Figura 49.

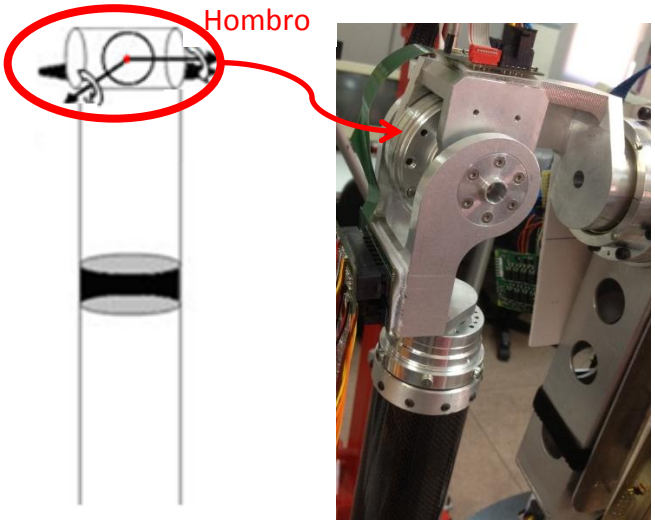

Figura 49: Grados de libertad del hombro.

La articulación cuyo eje es el X se utiliza para moverse el brazo hacia delante y hacia atrás, la articulación en el eje Z es para poder levantar la pierna hacia fuera.

5.3.2.2 Codo

El codo tiene dos grados de libertad y todos son rotatorios, sobre el eje X e Y, tal como se muestra en la Figura 50.

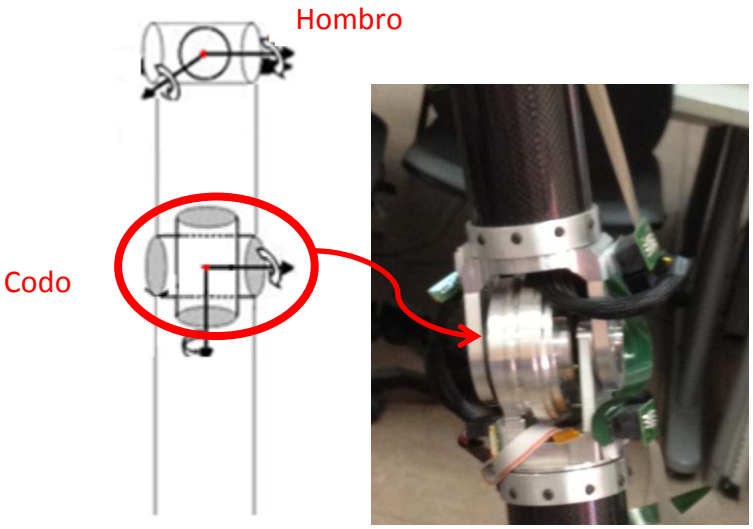

Figura 50: Grados de libertad del codo.

La articulación en el eje Y es para que el brazo pueda rotar sobre su propio eje, en el eje X es para levantar el brazo inferior.

5.3.2.3 Muñeca

La muñeca también tiene dos grados de libertad y todos son rotatorios, sobre el eje X e Y, tal como se muestra en la Figura 51.

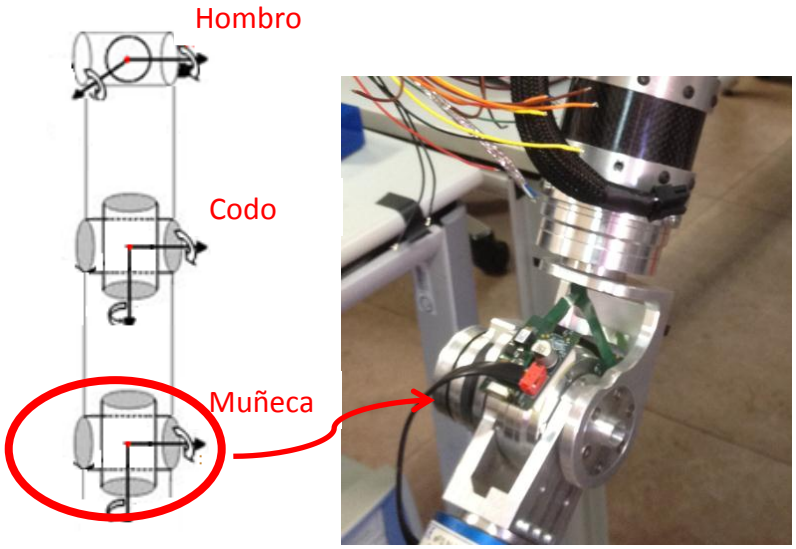

Figura 51: Grados de libertad de la muñeca.

La articulación en el eje Y es para que la mano pueda rotar sobre su propio eje, en el eje X es para levantar la mano.

5.3.3 Grados de libertad del tronco

El tronco posee 2 GDL, uno en el plano Y que le permite el giro del cuerpo sin tener que mover las piernas, y otro en el plano X que le permite regular su inclinación (ver la figura 52).

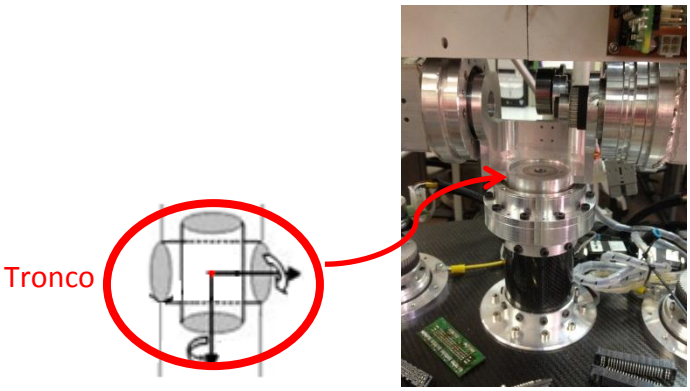

Figura 52: Grados de libertad del tronco.

5.3.4 Grados de libertad del cuello

El cuello posee 2 GDL, uno en el plano Y que le permite el giro la cabeza sin tener que mover todo el cuerpo, y otro en el plano X que le permite regular su inclinación (ver la figura 53).

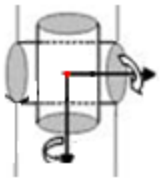

Figura 53: Grado de libertad del cuello.

A continuación en la figura 54 se muestran todas las articulaciones del modeloTEO.

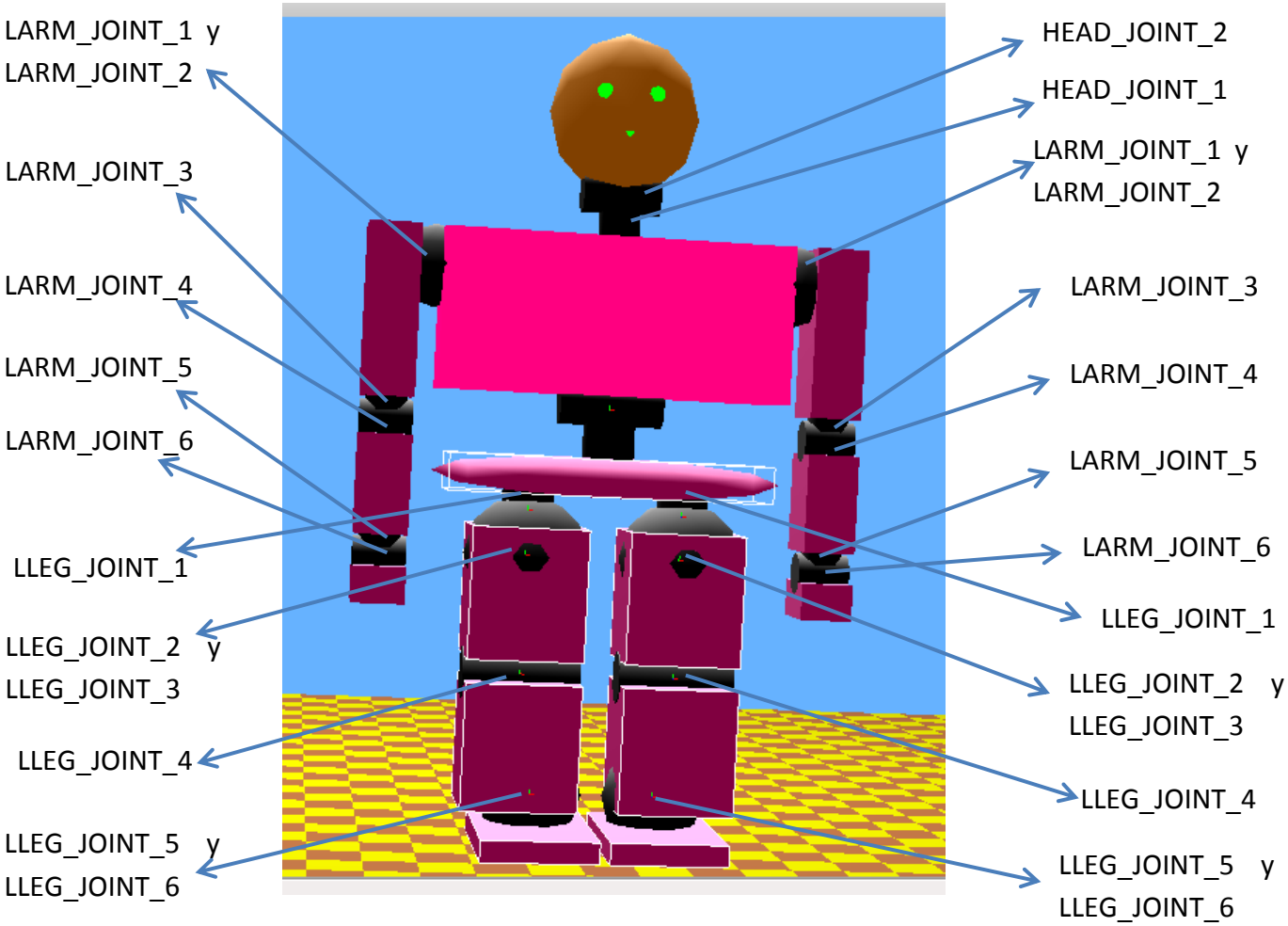

Figura 54: Todas las articulaciones del modeloTEO en Webots.

5.4 Eslabones del robot TEO

5.4.1 Cuerpo

El cuerpo de TEO consiste en dos nodos, el pecho y las dos articulaciones. El pecho está construido por un rectángulo de dimensión 0,676x0,305x0,15 m. Las dos articulaciones "BODY_JOINT_1" y "BODY_JOINT_2" están representadas por dos cilindros y sus medidas son:

"BODY_JOINT_1": h=0,18 m, r=0,05 m, rotación (X, Y, Z, ALFA): 0, 0 , 1, 1.57.

"BODY_JOINT_2": h=0,2m, r=0,04 m, rotación (X, Y, Z, ALFA): 0, 1, 0, 0.

5.4.2 Brazos

Cada brazo está dividido en 3 partes, la primera parte es brazo superior, la segunda parte es brazo inferior y la última es la mano del robot. Están construido por rectángulos y sus dimensiones son: 0,1x0,328x0,15, 0,1x0,19x0,15 y 0,1x0,07x0,15, respectivamente.

Las cuatro articulaciones "RARM_JOINT_1", "RARM_JOINT_2", "LARM_JOINT_1" y "LARM_JOINT_2" se representa por una esfera de radio 0,8 m.

Las articulaciones "RARM_JOINT_3", "RARM_JOINT_5", "LARM_JOINT_3" y "LARM_JOINT_5" están representadas por un cilindro y sus medidas son: h=0,1 m, r=0,04 m, rotación (X, Y, Z, ALFA): 0, 1 , 0, 0.

Las articulaciones "RARM_JOINT_4", "RARM_JOINT_6", "LARM_JOINT_4" y "LARM_JOINT_6" están representadas por un cilindro y sus medidas son: h=0,1 m, r=0,04 m, rotación (X, Y, Z, ALFA): 0, 0 , 1, 1,57.

5.4.3 Cadera

la cadera es de forma oval, para construir esta forma, el primer paso es crear una esfera de radio 0,075 m, y cambiando la escala a la siguiente medida(X, Y y Z): 2,26, 0,27, 1. La pierna del robot está formada por 3 partes, la pierna superior, la pierna inferior y el pie. Están construidas por rectángulos y sus dimensiones son: 0,22x0,25x0,22, 0,22x0,24x0,25 y 0,22x0,04x0,36, respectativamente.

Las dos primeras articulaciones son "RLEG_JOINT_1", "LLEG_JOINT_1", están representadas por un cilindro y sus medidas son: h=0,1 m, r=0,05 m, rotación (X, Y, Z, ALFA): 0, 1 , 0, 0.

Las cuatro articulaciones “RLEG_JOINT_2”, “RLEG_JOINT_3”, “LLEG_JOINT_2” y “LARM_JOINT_3” se representa por una esfera de radio 0,12 m.

Las articulaciones “RLEG_JOINT_4”, “LLEG_JOINT_4” están representadas por un cilindro y sus medidas son: h=0,225 m, r=0,05 m, rotación (X, Y, Z, ALFA): 0, 0, 1, 1,57.

Las cuatro articulaciones “RLEG_JOINT_5”, “RLEG_JOINT_6”, “LLEG_JOINT_6” y “LARM_JOINT_6” se representa por una esfera de radio 0,1 m.

5.4.4 Cabezas

El cuello tiene dos articulaciones “HEAD_JOINT_1” y “HEAD_JOINT_2”, están representadas por dos cilindros y sus medidas son:

“HEAD_JOINT_1”: h=0,1m, r=0,04 m, rotación (X, Y, Z, ALFA): 0, 1, 0, 0.

“HEAD_JOINT_2”: h=0,15 m, r=0,04m, rotación (X, Y, Z, ALFA): 0, 0 , 1, 1.57.

Y la cabeza está construida por una esfera, cuyo radio es 0,12 m, y con una escala de 01.12, 1.33, 0.8.

En la figura 55 se muestra el modelo del robot donde se señalan los eslabones que terminan de completar el modelo.

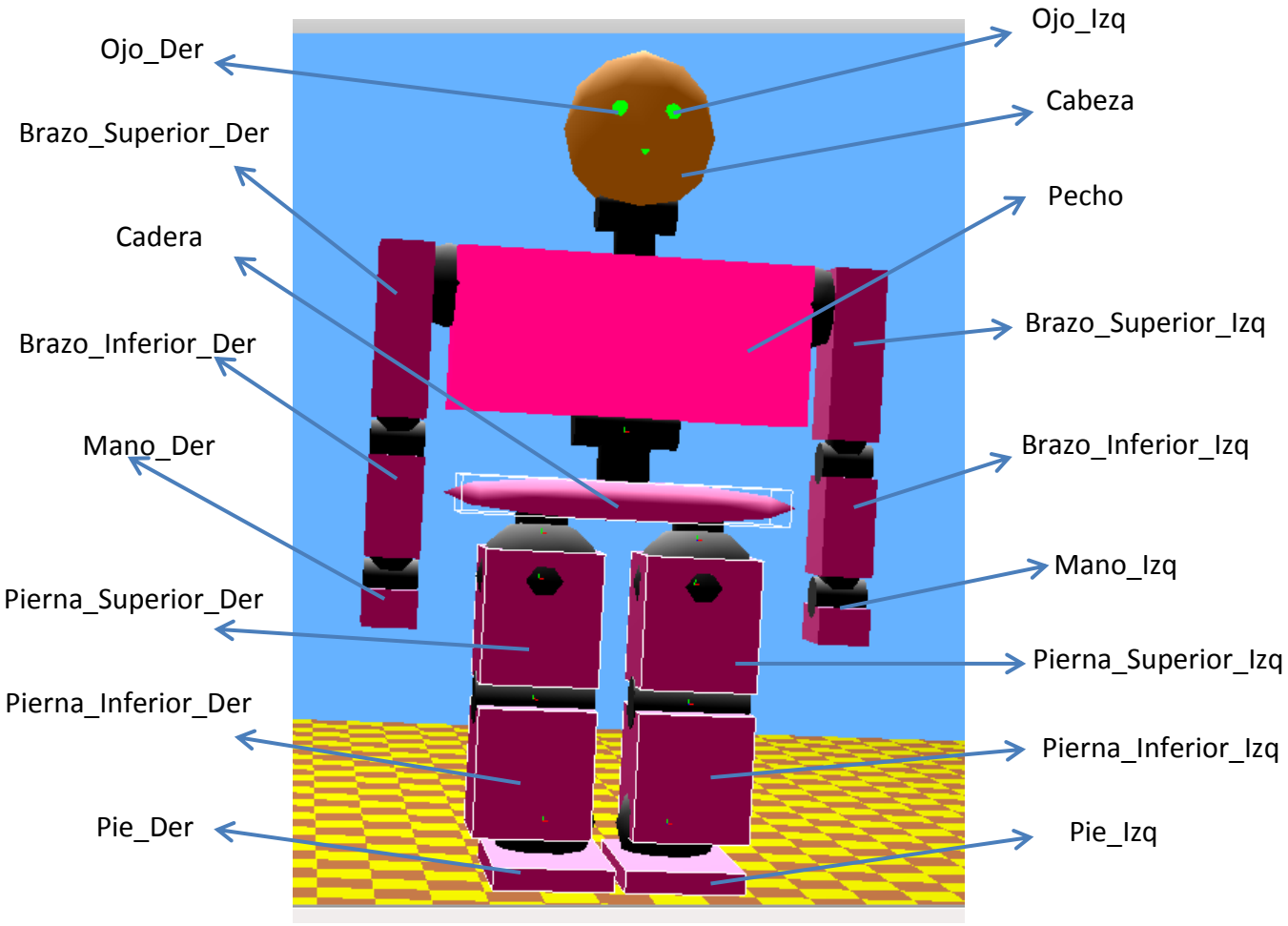

Figura 55: Modelo TEO con eslabones señalados en Webots.

6 Programación

En este capítulo programamos con el lenguaje C en el controlador. Para el primer paso, debemos asociar el controlador a nuestro robot TEO. Para asociarlo tenemos que ir a la ventana del árbol de escena del nodo *"TEORobot"*. Uno de sus atributos es *"controller" (ver la figura 56).* Si pulsamos sobre él, en la parte derecha de la ventana podemos pulsar sobre el botón de los puntos suspensivos para elegir el controlador a utilizar. [11]

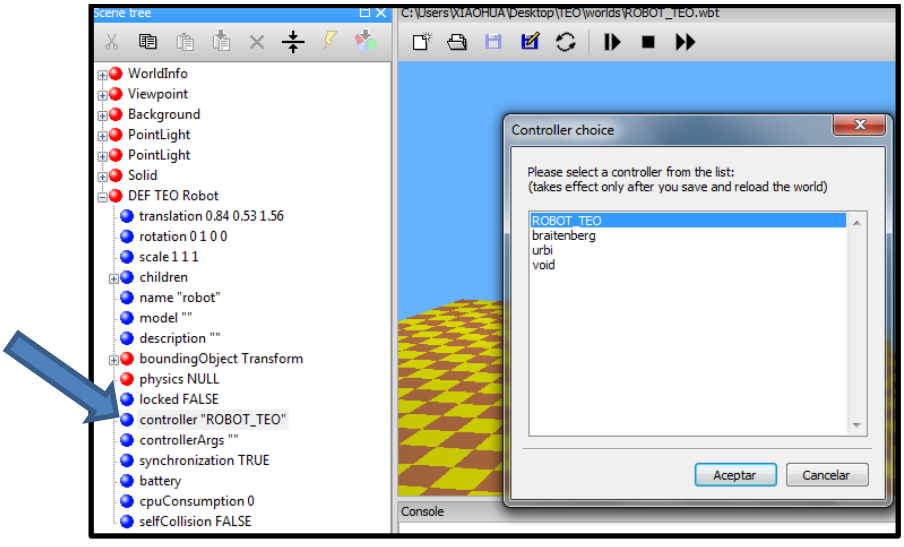

Figura 56: La ventana para asociar el controlador al robot en Webots.

Aparecerá una ventana en la derecha “Text Editor” (ver la figura 57) que contiene una serie de código en C.

```c
#include <webots/robot.h>

#define TIME_STEP 64


/*
 * This is the main program.
 * The arguments of the main function can be specified by the
 * "controllerArgs" field of the Robot node
 */
int main(int argc, char **argv)
{
  /* necessary to initialize webots stuff */
  wb_robot_init();

  /*
   * You should declare here DeviceTag variables for storing
   * robot devices like this:
   *  WbDeviceTag my_sensor = wb_robot_get_device("my_sensor");
   *  WbDeviceTag my_actuator = wb_robot_get_device("my_actuator");
   */

  /* main loop */
  do {

    /*
     * Enter here functions to send actuator commands, like:
     * wb_differential_wheels_set_speed(100.0,100.0);
     */

    /*
     * Perform a simulation step of 64 milliseconds
     * and leave the loop when the simulation is over
     */
  } while (wb_robot_step(TIME_STEP) != -1);

  /* Necessary to cleanup webots stuff */
  wb_robot_cleanup();

  return 0;
}
```

Figura 57: la estructura básica de un controlador para un robot wen Webots.

Este es una estructura básica del controlador, empieza con unas encabeceras de “.h” de lenguaje C, por ejemplo, “#include <Webot/robot.h>”. También es importante de definir la variable “TIME_STEP”, es el tiempo de muestreo.

A continuación, en la función “main”, funcion principal del controlador, es necesario hacer la inicialización del robot con la función “wb_robot_init()”.

Por último, se utiliza la función “wb_robot_cleanup()” para vaciar la memoria de webots.

6.1 Programación de secuencias

Nuestro robot consta de 28 GDL como ya hemos visto en la parte de la construcción del robot. Cada uno de ellos va a permitir al robot realizar diferentes movimientos. En este TFG intentamos programar sobre este robot para imitar algunos gestos sencillos del ser humano.

En primer lugar, hay que enumerar todas las articulaciones de nuestro robot para que nuestro controlador pueda reconocerlas y poder trabajar posteriormente con ellas (ver la figura 58). Estos son los nombres que utilizamos en el modelo. [10]

```
enum joints { BODY_JOINTS_1, BODY_JOINTS_2, HEAD_JOINTS_1,
HEAD_JOINTS_2, LLEG_JOINTS_1, LLEG_JOINTS_2, LLEG_JOINTS_3,
LLEG_JOINTS_4, LLEG_JOINTS_5, LLEG_JOINTS_6, RLEG_JOINTS_1,
RLEG_JOINTS_2, RLEG_JOINTS_3, RLEG_JOINTS_4, RLEG_JOINTS_5,
RLEG_JOINTS_6 };
```

Figure 58: Enumeración de las articulaciones.

Dentro de la función “main”, asignamos una variable a cada articulación. En la figura 59 utilizamos la función “WbDeviceTag” para definir una variable de tipo dispositivo o motor, a la que llamamos “servo_1”. La función wb_robot_get_device ("RLEG_JOINT_6") sirve para obtener la información del dispositivo o articulación “RLEG_JOINT_6”.

```
“WbDeviceTag servo_1 = wb_robot_get_device ("RLEG_JOINT_6")”
```

Figure 59: Ejemplo de asignación de una variable a la articulación “RLEG_JOINT_6”.

En la tabla 3 se muestra el código completo para asignar las variables a todas las articulaciones de nuestro robot TEO.

```
WbDeviceTag servo_1 = wb_robot_get_device ("RLEG_JOINT_6");
WbDeviceTag servo_2 = wb_robot_get_device ("RLEG_JOINT_5");
WbDeviceTag servo_3 = wb_robot_get_device ("RLEG_JOINT_4");
WbDeviceTag servo_4 = wb_robot_get_device ("RLEG_JOINT_3");
WbDeviceTag servo_5 = wb_robot_get_device ("RLEG_JOINT_2");
WbDeviceTag servo_6 = wb_robot_get_device ("RLEG_JOINT_1");
WbDeviceTag servo_7 = wb_robot_get_device ("LLEG_JOINT_1");
WbDeviceTag servo_8 = wb_robot_get_device ("LLEG_JOINT_2");
WbDeviceTag servo_9 = wb_robot_get_device ("LLEG_JOINT_3");
WbDeviceTag servo_10 = wb_robot_get_device ("LLEG_JOINT_4");
WbDeviceTag servo_11 = wb_robot_get_device ("LLEG_JOINT_5");
WbDeviceTag servo_12 = wb_robot_get_device ("LLEG_JOINT_6");
WbDeviceTag servo_13 = wb_robot_get_device ("BODY_JOINT_1");
WbDeviceTag servo_14 = wb_robot_get_device ("BODY_JOINT_2");
WbDeviceTag servo_15 = wb_robot_get_device ("RARM_JOINT_1");
WbDeviceTag servo_16 = wb_robot_get_device ("RARM_JOINT_2");
WbDeviceTag servo_17 = wb_robot_get_device ("RARM_JOINT_3");
WbDeviceTag servo_18 = wb_robot_get_device ("RARM_JOINT_4");
WbDeviceTag servo_19 = wb_robot_get_device ("RARM_JOINT_5");
WbDeviceTag servo_20 = wb_robot_get_device ("RARM_JOINT_6");
WbDeviceTag servo_21 = wb_robot_get_device ("LARM_JOINT_1");
WbDeviceTag servo_22 = wb_robot_get_device ("LARM_JOINT_2");
WbDeviceTag servo_23 = wb_robot_get_device ("LARM_JOINT_3");
WbDeviceTag servo_24 = wb_robot_get_device ("LARM_JOINT_4");
WbDeviceTag servo_25 = wb_robot_get_device ("LARM_JOINT_5");
WbDeviceTag servo_26 = wb_robot_get_device ("LARM_JOINT_6");
WbDeviceTag servo_27 = wb_robot_get_device ("HEAD_JOINT_1");
WbDeviceTag servo_28 = wb_robot_get_device ("HEAD_JOINT_2");
```

Tabla 3: El código completo para asiganar las variables a todas las articulaciones del robot.

6.2 Levantar los brazos

Levantar los dos brazos es un movimiento muy sencillo, simplemente hay que mover dos articulaciones de cada brazo: “RARM_JOINT_1”, “RARM_JOINT_4”, “LARM_JOINT_1”, “LARM_JOINT_4”.En la figura 60 se muestra el algoritmo de programación para levantar los brazos.

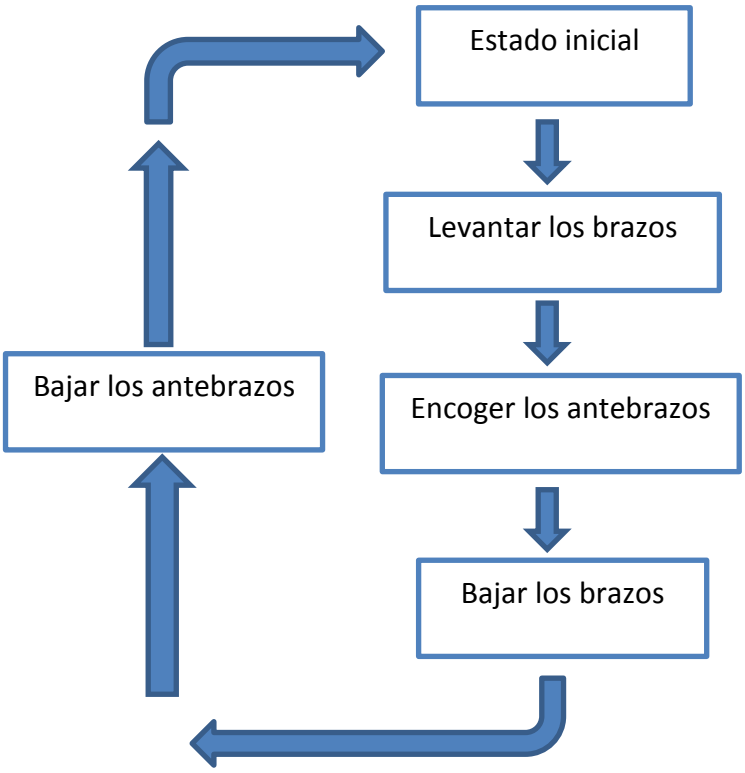

Figura 60: El algoritmo de programación para levantar los brazos

En la figura 61 se muestra el código para levantar los dos brazos al mismo tiempo. Como se puede observar, en este código solo utilizamos tres funcianes: la función “wb_servo_set_velocity()” , es para inicializar o ajustar una articulación determinada a una velocidad determinada, la unidad de la velocidad para la rotación es [radian/segundo], la función “wb_servo_set_position()”, es para indicar la posición final de una articulación, la unidad de desplazamiento de una rotación es [radian], y la función “wb_robot_step(TIME_STEP)” , es para ejecutar una etapa de simulación.

```
do {
  for (t=0; t<350; t++) // contar 350 pulsos
  {
    wb_servo_set_velocity(servo_15, 0.05);//ajustar velocidad de servo_15=0.05radian/s
    wb_servo_set_position(servo_15, -0.92);//indicar la posicion final del servo_15. -0

    wb_servo_set_velocity(servo_21, 0.05);//ajustar velocidad de servo_21=0.05radian/s
    wb_servo_set_position(servo_21, -0.92); //indicar la posicion final del servo_21
    wb_robot_step(TIME_STEP);              //Ejecutar una etapa de simulacion
  }
  for (t=0; t<350; t++) // contar 350 pulsos
  {
    wb_servo_set_velocity(servo_18, 0.05);//ajustar velocidad de servo_18=0.05radian/s
    wb_servo_set_position(servo_18, -1.5); //indicar la posicion final del servo_18

    wb_servo_set_velocity(servo_24, 0.05);//ajustar velocidad de servo_24=0.05radian/s
    wb_servo_set_position(servo_24, -1.5); //indicar la posicion final del servo_18
    wb_robot_step(TIME_STEP);
  }
  for (t=0; t<350; t++) // contar 350 pulsos
  {
    wb_servo_set_position(servo_15, 0); // vuelve a la posición inicial 0 radianes

    wb_servo_set_position(servo_21, 0); // vuelve a la posición inicial 0 radianes
    wb_robot_step(TIME_STEP);
  }
  for (t=0; t<450; t++) // contar 450 pulsos
  {
    wb_servo_set_position(servo_18, 0); // vuelve a la posición inicial 0 radianes

    wb_servo_set_position(servo_24, 0); //vuelve a la posición inicial 0 radianes
    wb_robot_step(TIME_STEP);
  }
  i++;
  } while (i<=10); // repite estas secuencias 11 veces
```

Figure 61: El código para levantar los dos brazos.

En este código utilizamos un bucle (do...While) para repetir el movimiento 11 veces. Dentro del bucle (do...While) hay otros cuatro bucles (for): el primer bucle “for” es para levantar los dos brazos enteros en 350 pulsos y a un ángulo -0.92 radianes con respecto a su posición inicial, una vez haya terminado el primer movimiento, se ejecuta el siguiente “for”, este bucle manda una orden de girar sus codos a un ángulo de -1.5 radianes con respecto a su posición inicial, el tercer y cuarto bucle son para volver a sus posiciones iniciales. Todas sus velocidades son 0.05 radian/segundo.

En la figura 62 se muestran las secuencias de levantar los brazos de modelo en Webots. El robot se encuentra en estado de reposo después levantan sus brazos estirados, en tercer lugar encoge los antebrazos para luego bajar sus brazos, y por último, baja sus antebrazos y vuelve al estado inicial.

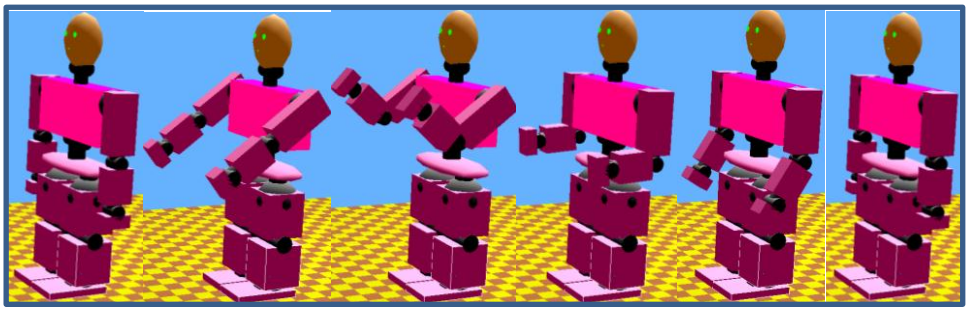

Figure 62: Las secuencias de levantar los brazos.

6.3 Imitar el caminar

Imitar el caminar es el resultado de mover dos articulaciones de cada pierna: “RLEG_JOINT_3”, “RLEG_JOINT_4”, “LLEG_JOINT_3” y “LLEG_JOINT_4”. Para que la forma de caminata tenga la apariencia de ser humano, necesitamos que se muevan también los dos brazos, las articulaciones correspondientes son : “LARM_JOINT_1” y “RARM_JOINT_1”. En la figura 63 se muestra el algoritmo de programación.

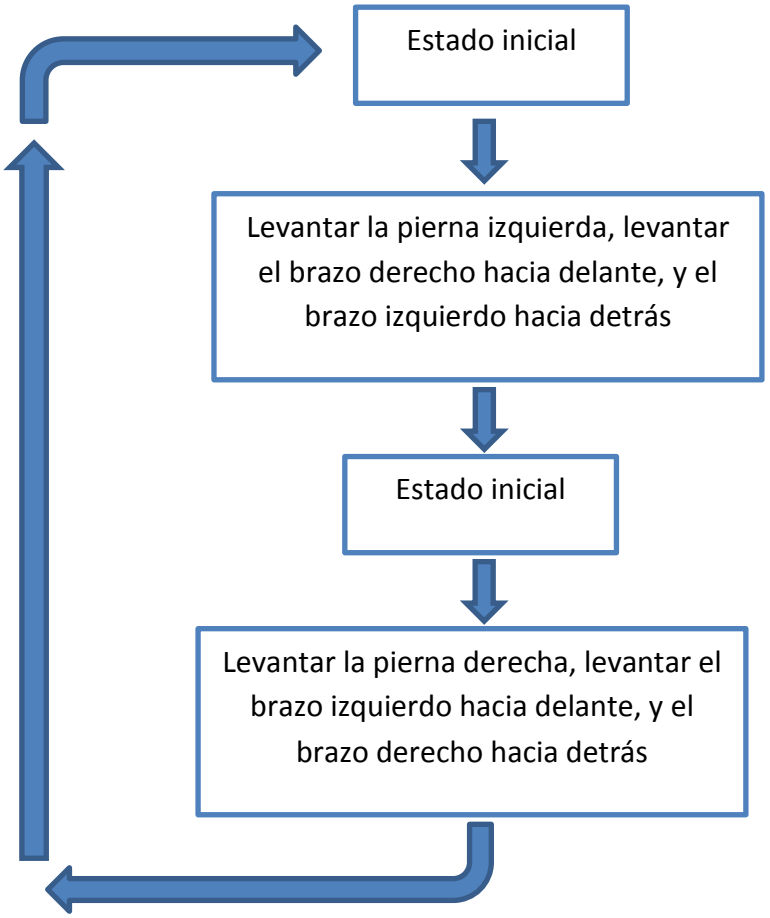

Figura 63: El algoritmo de progamación de imitar el caminar.

En la figura 64 se muestra el código para hacer la imitación de la camitata de ser humano. En este código utilizamos un bucle (do...While) para repetir el movimiento en 11 veces. Dentro de este bucle cerrado contiene otro cuatro bucles (for): el primer bucle “for” es para definir la velocidad y el ángulo hay que girar las seis articulaciones en un ritmo de 350 pulsos, el segundo “for” es regresar a sus posiciones iniciales, el tercer “for” es idéntico q que el primero, simplemente cambia el sentido de giro de todas sus articulaciones, y por último, se regrasan todas las articulaciones a sus estados iniciales.

```
do {
  for (t=0; t<350; t++){ // contar 350 pulsos
    wb_servo_set_velocity(servo_15, 0.05);//ajustar velocidad de servo_15=0.05radian/s
    wb_servo_set_position(servo_15, 0.92);//indicar la posicion final del servo_15. 0.9
    wb_servo_set_velocity(servo_21, 0.05);//ajustar velocidad de servo_21=0.05radian/s
    wb_servo_set_position(servo_21, -0.92);//indicar la posicion final del servo_21. -0
    wb_servo_set_velocity(servo_4, 0.05);//ajustar velocidad de servo_4=0.05radian/s
    wb_servo_set_position(servo_4, -0.92);//indicar la posicion final del servo_4. -0.9
    wb_servo_set_velocity(servo_3, 0.05);//ajustar velocidad de servo_3=0.05radian/s
    wb_servo_set_position(servo_3, 0.92);//indicar la posicion final del servo_3. 0.92
    wb_robot_step(TIME_STEP);          //Ejecutar una etapa de simulacion
  }
  for (t=0; t<350; t++){ // contar 350 pulsos
    wb_servo_set_position(servo_15, 0);// vuelve a la posición inicial 0 radianes
    wb_servo_set_position(servo_21, 0);// vuelve a la posición inicial 0 radianes
    wb_servo_set_position(servo_4, 0);// vuelve a la posición inicial 0 radianes
    wb_servo_set_position(servo_3, 0);// vuelve a la posición inicial 0 radianes
    wb_robot_step(TIME_STEP);
  }
  for (t=0; t<350; t++){ // contar 350 pulsos
    wb_servo_set_velocity(servo_15, 0.05);//ajustar velocidad de servo_15=0.05radian/s
    wb_servo_set_position(servo_15, -0.92);//indicar la posicion final del servo_15.
    wb_servo_set_velocity(servo_21, 0.05);//ajustar velocidad de servo_21=0.05radian/s
    wb_servo_set_position(servo_21, 0.92);//indicar la posicion final del servo_21.
    wb_servo_set_velocity(servo_9, 0.05);//ajustar velocidad de servo_9=0.05radian/s
    wb_servo_set_position(servo_9, -0.92);//indicar la posicion final del servo_9.
    wb_servo_set_velocity(servo_10, 0.05);//ajustar velocidad de servo_10=0.05radian/s
    wb_servo_set_position(servo_10, 0.92);//indicar la posicion final del servo_10.
    wb_robot_step(TIME_STEP);
  }
  for (t=0; t<350; t++){ // contar 350 pulsos
    wb_servo_set_position(servo_15, 0);// vuelve a la posición inicial 0 radianes
    wb_servo_set_position(servo_21, 0);// vuelve a la posición inicial 0 radianes
    wb_servo_set_position(servo_9, 0);// vuelve a la posición inicial 0 radianes
    wb_servo_set_position(servo_10, 0);// vuelve a la posición inicial 0 radianes
    wb_robot_step(TIME_STEP);
  }
  i++;
  } while (i<=10); // repite estas secuencias 11 veces
```

Figure 64: El código para hacer la imitación de la camitata de ser humano.

En la figura 65 se muestra las secuencias de la caminata de modelo en Webots. Al principio, el robot se encuentra en estado reposo, a continiación se mueven su pierna izquierda y su brazo derecha hacia delante, y su brazo izquierda se mueve al sentido contrario. Una vez hayan llegado las posiciones finales, se vuelven al estado inicial, a continiación se repite el mismo movimiento que anteriores pero con otra pierna y los brazos cambian el sentido de giro.

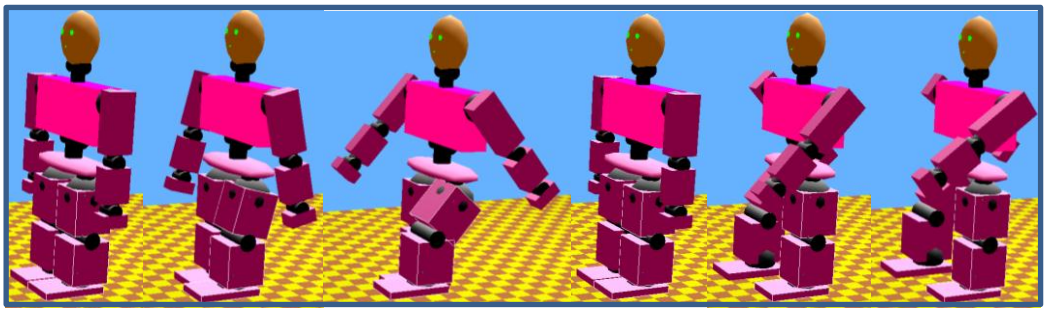

Figure 65: Las secuencias de la caminata de modelo en Webots.

6.4 Inclinar el cuerpo

Para inclinar el cuerpo hemos empleado el movimiento de cinco articulaciones: “RARM_JOINT_2”, “LARM_JOINT_2”, “BODY_JOINT_2”, “LLEG_JOINT_3”, “LLEG_JOINT_4”. En la figura 66 se muestra el algoritmo de programación para inclinar el cuerpo.

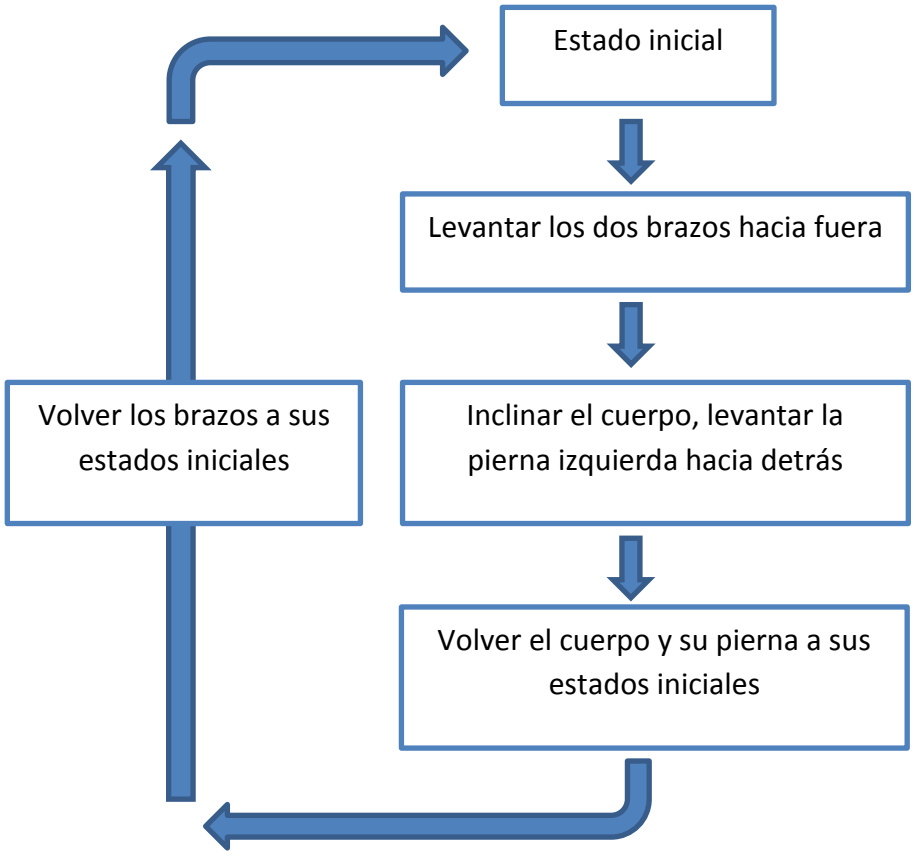

Figura 66: El algoritmo de programación para inclinar el cuerpo

En la figura 67 se muestra el código de la inclinación del cuerpo. En este código utilizamos un bucle (do...While) para repetir el movimiento 11 veces. Este bucle cerrado contiene otro cuatro bucles (for): el primer bucle “for” es para definir la velocidad y el ángulo que tienen que girar los brazos al elevarse, el segundo “for” es para levantar la pierna izquierda y poder inclinar el cuerpo hacia delante, el tercer “for” es para poner a su estado inicial el cuerpo y la pierna, y con el último “for”, regrasan los brazo a sus estados iniciales.

```
do {

  for (t=0; t<400; t++) // contar 400 pulsos
  {
    wb_servo_set_velocity(servo_16, 0.05);//ajustar velocidad de servo_16=0.05radian/s
    wb_servo_set_position(servo_16, -1.2);//indicar la posicion final del servo_15. -1.
    wb_servo_set_velocity(servo_22, 0.05);//ajustar velocidad de servo_16=0.05radian/s
    wb_servo_set_position(servo_22, 1.2);//indicar la posicion final del servo_15. 1.2r
    wb_robot_step(TIME_STEP);       //Ejecutar una etapa de simulacion
  }
  for (t=0; t<400; t++) // contar 400 pulsos
  {
    wb_servo_set_velocity(servo_9, 0.03);//ajustar velocidad de servo_9=0.03radian/s
    wb_servo_set_position(servo_9, 0.5);//indicar la posicion final del servo_9. 0.5rad
    wb_servo_set_velocity(servo_10, 0.02);//ajustar velocidad de servo_10=0.02radian/s
    wb_servo_set_position(servo_10, 1.2); //indicar la posicion final del servo_10. 1.2
    wb_servo_set_velocity(servo_14, 0.03);//ajustar velocidad de servo_114=0.03radian/s
    wb_servo_set_position(servo_14, 0.8);//indicar la posicion final del servo_14. 0.8r
    wb_robot_step(TIME_STEP);
  }

  for (t=0; t<400; t++) // contar 400 pulsos
  {
    wb_servo_set_position(servo_9, 0);// vuelve a la posición inicial 0 radianes
    wb_servo_set_position(servo_10, 0);// vuelve a la posición inicial 0 radianes
    wb_servo_set_position(servo_14, 0);// vuelve a la posición inicial 0 radianes
    wb_robot_step(TIME_STEP);
  }
  for (t=0; t<400; t++) // contar 400 pulsos
  {
    wb_servo_set_position(servo_16, 0);// vuelve a la posición inicial 0 radianes
    wb_servo_set_position(servo_22, 0);// vuelve a la posición inicial 0 radianes
    wb_robot_step(TIME_STEP);
  }
  i++;
  } while (i<=10); // repite estas secuencias 11 veces
```

Figura 67: El código para hacer la inclinación del cuerpo.

En la figura 68 se muestran las secuencias de la inclinación del cuerpo en Webots. Al principio, el robot se encuentra en estado de reposo, la acción siguiente que realiza el robot es levantar los brazos estirados en arco, una vez levantados los brazo conjuntamente levanta la pierna izquierda hacia detrás e inclina el cuerpo hacia delante. Una vez que llega a las posiciones finales, tanto el cuerpo como la pierna vuelven a su estado inicial, por último los brazos se encogen hasta llegar al estado de reposo.

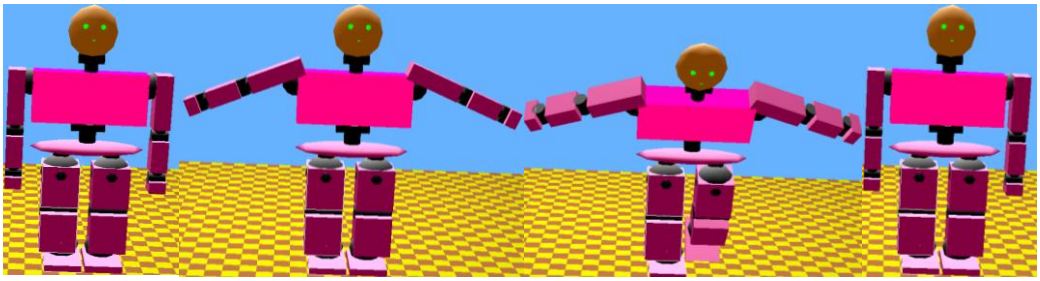

Figura 68: Las secuencias de la inclinación del cuerpo en Webots

7 Conclusiones

El principal objetivo de este proyecto consiste en el modelado de la plataforma robótica TEO para el simulador Webots. Dicho modelado permite la simulación de tareas simples del robot TEO en el entorno virtual. Para ello ha sido necesario familiarizarse con el entorno de trabajo por medio de un estudio detallado del simulador Webots y así poder comprender todo su funcionamiento. Como apoyo se tienen 2 documentos: la guía de referencia y el manual de usuario. El programa también tiene varios ejemplos que se fueron consultando y realizando progresivamente añadiendo características al robot creado.

Se ha desarrollado el modelo VRML del robot TEO para la plataforma Webots. Este modelo se ha desarrollado siguiendo las características cinemáticas del robot humanoide TEO.

Para construir dicho modelo, hay que conocer sus nodos y el funcionamiento de los nuevos nodos que incorpora Webots para el robot TEO, para ello, diseñamos las estructuras jerárquica de los nodos del robot TEO. Debido al movimiento de todas las articulaciones de robot son rotacional, es imprescindible conocer sus ejes de rotación, nos facilita la programación posteriormente. También se incluyen todas las medidas originales de cada una de las piezas que forman el robot para conseguir un modelo lo más fiel al robot real.

Una vez desarrollado el modelo, mediante el simulador, se ha programado en lenguaje C para realizar varias tareas, como levantar los brazos, imitar el caminar del ser humano e inclinar el cuerpo. Para ello se ha estudiado la dirección y el sentido de

todas las articulaciones del robot, y posteriormente se ha construido un esquema del algoritmo para cada movimiento.

Finalmente se ha grabado varios videos de los movimientos que hemos simulado con el modelo de robot TEO en Webots.

En conclusión, se pueden considerar los objetivos del trabajo fin de grado propuesto como conseguidos.

8 Trabajos futuros

Como en todo proyecto existen una serie de cuestiones dignas de ser estudiadas para una posible ampliación. A continuación se hablará de las que quizá deberían considerarse prioritarias, ya sea por la mejora que supone al proyecto o bien por su utilidad posterior.

En primer lugar, me gustaría que el robot pudiera hacer tareas o movimientos más complejos que los que hemos simulado en este proyecto,

Además, el robot TEO dispone de una serie de sensores que sería conveniente añadir también al modelo virtual. Estos sensores son:

- El sensor inercial (IMU): para medir la aceleración y la velocidad angular. Se utiliza para análisis de movimiento y control del equilibrio.
- El sensor FUERZA/PAR: Los robots requieren de este tipo de sensores para calcular, entre otros parámetros, el ZMP, información que durante la caminata será utilizada para ajustar la posición de los actuadores encargados del equilibrio.
- Una cámara de alta resolución (KINECT) para analizar visualmente el entorno. Kinect cuenta con una cámara de video a tres colores y sensor de profundidad que es capaz de reconocer la cara humana y ver el espacio en tres dimensiones bajo cualquiera condición de iluminación.

Debido a la falta de datos y que es difícil de conseguirlos de forma exacta, no hemos incluido la masa de los eslabones de nuestro robot TEO. Por dicha razón, nuestro modelo no puede desplazarse de un sitio a otro.

No obstante, podríamos hacer las estimaciones de la masa total del robot y sus eslabones. Una vez el robot tenga masa será imprescindible colocar un controlador de equilibrio, para evitar la caída del robot.

Otro aspecto a tener en cuenta sería la auto-colisión. En el mundo virtual el modelo del robot puede girar sus articulaciones sin limitaciones, por ejemplo virtualmente las articulaciones pueden coincidir en el mismo punto sin colisionar y en el mundo real eso sería imposible.

Por último, para mejorar el modelo del robot, deberíamos poder modificar los eslabones interpretados por figuras geométricas, por eslabones más complejos y con apariencia física al cuerpo humano. Para más realismo natural, se pueden crear entornos para la simulación, como por ejemplo el salón de una casa, el campo de fútbol o la calle de una ciudad.

9 Bibliografía

9.1 Páginas Web:

[1] Robótica

http://es.wikipedia.org/wiki/Robots (Julio de 2011)

[2] Clasificación de los robots

http://usuarios.lycos.es/sparta/experiences12.html (Julio de 2012)

[3] SimRobot

http://www.informatik.uni-bremen.de/simrobot/index_e.htm (Julio de 2012)

[4] Gazebo

http://gazebosim.org/ (Julio de 2012)

[5] OpenHRP3

http://www.openrtp.jp/openhrp3/en/about.html (Julio de 2012)

[6] Marilou Robotics Studio

http://www.anykode.com/index.php (Julio de 2012)

[7] Microsoft Robotics Developer Studio 4

http://es.wikipedia.org/wiki/Microsoft_Robotics_Studio (Julio de 2012)

[8] OpenRAVE

http://en.wikipedia.org/wiki/OpenRAVE (Julio de 2012)

[9] VRML

http://es.wikipedia.org/wiki/VRML (Agosto de 2012)

[10] Webots User Guide (release 6.4.4) copyright (c) 2011 Cyberbotics Ltd. All rights reserved.

http://www.cyberbotics.com (la última vez 03 de septiempre de 2012)

[11] Webots Reference Manual (release 6.4.4) copyright (c) 2011 Cyberbotics Ltd. All rights reserved.

http://www.cyberbotics.com (la última vez 03 de septiempre de 2012)